Titre: Title:	Parallélisation des simulations du logiciel EMTP en utilisant des sémaphores et le standard FMI
Auteur: Author:	Sara Montplaisir-Gonçalves
Date:	2015
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Montplaisir-Gonçalves, S. (2015). Parallélisation des simulations du logiciel EMTP en utilisant des sémaphores et le standard FMI [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1687/

UNIVERSITÉ DE MONTRÉAL

PARALLÉLISATION DES SIMULATIONS DU LOGICIEL EMTP EN UTILISANT DES SÉMAPHORES ET LE STANDARD FMI

SARA MONTPLAISIR-GONÇALVES

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

MARS 2015

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

PARALLÉLISATION DES SIMULATIONS DU LOGICIEL EMTP EN UTILISANT DES SÉMAPHORES ET LE STANDARD FMI

présenté par : MONTPLAISIR-GONÇALVES Sara

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. KARIMI Houshang, Ph. D., président

M. MAHSEREDJIAN Jean, Ph. D., membre et directeur de recherche

M. SAAD Omar, M. Sc. A., membre et codirecteur de recherche

M. KOCAR Ilhan, Ph. D., membre

DÉDICACE

À ma mère,

qui m'a transmis
son amour des mots,
le plaisir de la lecture,
et l'importance du savoir.

REMERCIEMENTS

Ce projet a été rendu possible grâce à l'apport de plusieurs personnes. J'aimerais prendre quelques instants pour souligner leur contribution.

Je voudrais tout d'abord adresser mes remerciements à mon directeur de recherche Dr. Jean Mahseredjian, qui m'a introduite au monde de la recherche et qui m'a inspirée à poursuivre des études supérieures à l'École Polytechnique de Montréal.

Je dois également remercier M. Omar Saad, chercheur à l'Institut de recherche d'Hydro-Québec et co-directeur de mon projet. Je le remercie pour l'aide incommensurable qu'il m'a apportée dans les moments les plus critiques de mon projet.

Je voudrais ensuite souligner le support de la part de mes gestionnaires chez Hydro-Québec, MM. François Lévesque et Pascal Prud'homme. Merci d'avoir cru en mon travail et dans les retombées positives de ma recherche pour l'entreprise.

Je tiens aussi à remercier MM. Xavier Legrand et Ali El-Akoum, ingénieurs chez EDF, qui m'ont apporté tout le support nécessaire dans les débuts de mon projet avec l'outil qui fut la base fondatrice des travaux présentés ici.

Puis, je dois remercier ma famille et mes amis qui m'ont soutenue pendant ces moments parfois périlleux, tout spécialement Simon pour sa rigueur et sa grande disponibilité. J'aimerais aussi souligner l'aide et l'amitié de tous les étudiants gradués de la section « Énergie » du département de génie électrique. Vos conseils et votre présence ont été très appréciés.

Finalement, merci à vous, membres du jury et autres lecteurs, pour l'intérêt que vous portez à ce projet de recherche. J'espère rendre cette lecture aussi intéressante que cette aventure l'a été pour moi.

RÉSUMÉ

Ce mémoire présente un outil de co-simulation utilisé pour lier plusieurs instances du logiciel *Electromagnetic transients program* (EMTP, version RV) à travers une ou plusieurs lignes (ou câbles) de transmission en parallèle. Le délai de propagation intrinsèque des lignes de transmission en font des points de découplage naturels d'un grand réseau de transmission en plusieurs sous-réseaux plus petits. Ces sous-réseaux peuvent ainsi être résolus en parallèle à chaque pas de temps et sans approximation dans les calculs. Un processus dit de *co-simulation* est alors établi, où chaque instance d'EMTP résolvant un sous-réseau est traitée par une unité de calcul (cœur) distincte et partage à chaque pas de temps les informations requises à travers la ligne de transmission de découplage. Ce type de parallélisation permet une réduction du temps de simulation d'un réseau donné. L'outil présenté ici permet une séparation arbitraire gérée par l'utilisateur, mais les gains de performance dépendent de la grandeur et des modèles contenus dans les sous-réseaux. Cet aspect est analysé dans le mémoire. D'autres aspects, tels que les délais de communication, sont également abordés.

L'outil de co-simulation présenté est développé selon une architecture extensible et permet donc d'ajouter des modèles ou des extensions aux méthodes actuelles. Les fichiers sources responsables du partage des données sont basés sur la norme européenne *Functional Mock-Up Interface* (FMI), utilisée pour la création d'une interface de co-simulation entre logiciels de simulation selon un schéma de communication maître-esclave(s).

La méthode de partage de données a été implémentée en utilisant des primitives de synchronisation de bas niveau appelées *sémaphores*. Les sémaphores sont largement utilisés dans les programmes informatiques pour résoudre des situations de concurrence critique lors de partage de données entre plusieurs programmes ou fils d'exécution. Ils sont utilisés autant pour verrouiller l'accès aux données utilisées par un fil d'exécution que comme variable de compétition (en anglais, *condition variable*), afin de gérer et de protéger les opérations de lecture et d'écriture des variables partagées. Dans cet outil, plusieurs sémaphores sont utilisés pour implémenter les opérations d'initialisation et de communication du maître et des esclaves.

L'approche présentée est testée sur des réseaux de référence réels. Les réseaux soumis peuvent être utilisés par la communauté scientifique pour vérifier les gains obtenus par la méthode de parallélisation utilisée dans ce mémoire et d'autres approches. Plusieurs de ces réseaux ont amené

des gains notables de performance lors de la parallélisation. Deux cas, soit une variante des réseaux IEEE-39 barres et un réseau réel de grandes dimensions, ont donné des améliorations du temps de simulation de plus de 40% sur un ordinateur de travail standard à quatre cœurs. Les réseaux de référence présentés comportent des modèles détaillés de machines synchrones avec régulation de tension (excitatrice) et régulation de vitesse (gouverneur), des modèles de transformateurs avec données de magnétisation du noyau, des modèles exponentiels de charge et des modèles de lignes de transmission à paramètres constants.

ABSTRACT

This thesis presents a co-simulation tool used to link multiple instances of the Electromagnetic Transients Program (EMTP, RV version) through one or more transmission lines or cables. The intrinsic propagation delays of transmission lines allow to naturally decouple a large network into several separate subnetworks. The subnetworks can then be solved in parallel and without approximations. A co-simulation process is established, where each instance of EMTP running on a separate processing unit, solves its own subnetwork and shares data through linking transmission lines at every time-step. The resulting parallelization allows reducing the computing times for the total simulation process. With this approach, the network can be separated at arbitrary locations, but the computing time gains are dependent on the size and contents of the resulting subnetworks. This aspect is analyzed in this thesis through the presented benchmarks. Other aspects, such as communication delays between decoupled networks, are also studied.

The presented co-simulation tool is developed using an extensible framework that allows easily adding models and extensions to available methods. The tool is compliant with the Functional Mock-up Interface (FMI) co-simulation standard that is used for coupling two or more simulation tools in a co-simulation environment using a master-slave communication method. The FMI standard allows extending the co-simulation setup to include other simulation packages.

The co-simulation (sharing) method has been implemented using low-level synchronization primitives known as semaphores. They are widely used in computer programming to solve concurrency problems where data is shared between multiple programs or threads. They are used both as locks and as condition variables to protect and manage read/write operations on shared variables. For this tool, a number of binary semaphores have been implemented for initialization, and master-slave operations.

The presented approach is tested with practical power system benchmarks. The contributed benchmarks can be used for testing parallelization gains on practical power grids. Many of the test cases used in this project exhibited notable performance gains when using the co-simulation tool for parallelization of the simulation. In some cases, for the IEEE-39 bus system and for a practical power system, on a standard quad-core computer, this approach reduced computing times by more than 40%. The presented systems include detailed synchronous generator models

with exciter and governor controls, transformers with magnetization data, exponential load models, and constant-parameter line models.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

API	*Application Programming Interface* (interface de programmation)
CP	*Constant Parameters* (ligne à paramètres constants)
CPU	*Central Processing Unit* (unité centrale de traitement de données)
DLL	*Dynamic-link Library* (bibliothèque de liens dynamiques)
EDF	Électricité de France
EMT	*Electromagnetic Transient* (transitoires électromagnétiques)
EMTP-RV	*Electromagnetic Transient Program – Restructured Version*
FD	*Frequency Dependent* (ligne à paramètres dépendant de la fréquence)
FMI	*Functional Mock-up Interface*
GPU	*Graphics Processing Unit* (processeur graphique)
HVDC	*High Voltage Direct Current* (courant continu à haute tension)
IEEE	*Institute of Electrical and Electronics Engineers*
MMC	*Modular Multi-Level Converter* (convertisseur multi-niveaux)
MMF	*Memory-mapped file* (fichier mappé en mémoire)
MPI	*Message Passing Interface*
NUMA	*Non Uniform Memory Access* (UCT en accès mémoire non-uniforme)
RAM	*Random-access memory* (mémoire vive)
TCP/IP	*Transmission Control Protocol / Internet Protocol*
UCT	Unité centrale de traitement de données

CHAPITRE 1 INTRODUCTION

1.1 Description du problème et contexte

L'étude des grands réseaux électriques en vue de leur opération et de leur amélioration est un important défi du 21^{e} siècle. En effet, ces réseaux possèdent des conditions d'opération particulières étant donné trois principales raisons. Tout d'abord, la fiabilité est le critère le plus important pour leur fonctionnement étant donné les importantes pertes économiques et l'impact social négatif qu'engendrent les pannes. Puis, les coûts de construction et d'opération d'un réseau électrique sont très élevés.

Les réseaux électriques modernes sont de plus en plus complexes et comprennent maintenant des diversifications avec l'utilisation croissante de transmission à courant continu et des sources d'énergie éolienne. La robustesse et la fiabilité des réseaux nécessitent des études très détaillées en utilisant des outils de simulation. Plusieurs outils et méthodes de simulation sont disponibles selon les besoins de précision et parfois selon les données disponibles. L'approche la plus précise utilisée présentement est basée sur une représentation par éléments de circuit électrique et par simulation directe dans le domaine du temps sans limitations aux niveaux de la topologie et des modèles employés. Cette approche, de type large bande de fréquences, est appelée simulation des transitoires électromagnétiques. Le logiciel visé dans ce mémoire porte le nom EMTP (ElectroMagnetic Transients Program, version RV) [1], [2], [3].

Une difficulté importante avec EMTP est au niveau des temps de calcul requis pour la simulation détaillée des réseaux de grandes dimensions (un grand nombre de composants et/ou de nœuds électriques). Ce mémoire propose une approche de parallélisation du logiciel EMTP, basée sur la co-simulation.

La version actuelle du logiciel est programmée de façon séquentielle. Les équations des modèles sont assemblées et insérées dans un système d'équations global solutionné à chaque pas de temps. La solution du système d'équations global est suivie par la mise à jour des modèles pour la solution subséquente. Toutes les opérations sont effectuées de façon séquentielle et sur un seul cœur (processeur) de l'ordinateur. Cette approche n'arrive pas à bénéficier des systèmes multi-cœurs disponibles présentement. De plus, pour la technologie informatique actuelle, la parallélisation demeure la seule façon d'accélérer les calculs, puisque la vitesse des processeurs

individuels a atteint des niveaux de saturation. Il est donc nécessaire de travailler sur la parallélisation des calculs d'EMTP en utilisant des plateformes de type multi-cœurs.

1.2 Objectifs de recherche

L'objectif général du projet de recherche est d'utiliser les unités de calcul parallèles des ordinateurs personnels lors des simulations temporelles du logiciel EMTP afin d'augmenter l'efficacité du moteur de calcul et ainsi diminuer le temps de simulation.

Pour atteindre l'objectif principal du projet, les quatre objectifs spécifiques suivants ont été déterminés :

1. Étudier les approches de programmation parallèle pour des applications de type EMTP;
2. Choisir une approche de parallélisation éprouvée et livrable dans le cadre d'un projet de maîtrise;
3. Développer et utiliser des cas de tests pour valider l'approche de parallélisation et analyser les gains qui en résultent;
4. Proposer une preuve de concept d'une méthode de résolution en parallèle en démontrant les gains en temps de calcul avec des cas pratiques.

1.3 Plan du mémoire

Le Chapitre 2 présente la revue de littérature effectuée dans le cadre de ce projet. La revue est ciblée pour les besoins des travaux effectués dans ce projet.

Le Chapitre 3 détaille l'implémentation de la solution de parallélisation proposée. Le fonctionnement de l'outil développé ainsi que son intégration au logiciel de simulation sont expliqués en détails.

Puis, le Chapitre 4 porte sur la présentation et l'analyse des résultats. Les cas de tests utilisés sont présentés et la validation de l'outil est réalisée en comparant avec un cas de référence sans parallélisation. Les améliorations de performance sont alors présentées et analysées pour chaque cas de test.

Finalement, la conclusion présente une synthèse des travaux réalisés et identifie les prochaines étapes d'amélioration de l'outil actuel.

1.4 Contributions originales

Plusieurs contributions originales ont été réalisées dans le cadre de ce projet afin de proposer un outil fonctionnel de parallélisation du logiciel EMTP :

1. Initialisation précise des sous-réseaux en utilisant la solution en régime permanent du réseau complet;
2. Possibilité d'avoir plusieurs lignes de transmission entre un couple de co-simulation maître-esclave;
3. Proposition d'un outil testé à l'aide de réseaux réels, et des travaux qui proposent des réseaux de référence réalistes pour le champ d'application;
4. Travail réalisé dans le cadre d'un logiciel qui solutionne les réseaux avec des algorithmes de matrices creuse, permettant des gains supplémentaires.

CHAPITRE 2 REVUE DE LITTÉRATURE

La présente revue de littérature est divisée en trois sections dont chacune couvre un aspect distinct de la recherche nécessaire à la parallélisation d'un logiciel de simulation de transitoires électromagnétiques. La première section présente brièvement la méthode de simulation des réseaux électriques par ordinateur, plus particulièrement des transitoires électromagnétiques. La seconde section fait un rappel sur la modélisation des lignes de transmission électrique. La dernière section fait un survol de l'architecture multi-cœurs des ordinateurs modernes et énumère certaines stratégies de parallélisation proposées dans la littérature.

2.1 Simulation des réseaux électriques

Les logiciels de type EMT sont conçus afin d'étudier les transitoires électromagnétiques, mais s'appliquent aussi à la simulation des phénomènes plus lents, comme, par exemple, les oscillations électromécaniques d'un réseau. EMTP est un logiciel de type large bande de fréquences et permet, mathématiquement parlant, de simuler des phénomènes lents et rapides dans un réseau électriques. EMTP est basé sur la représentation du réseau par un circuit électrique équivalent tout en minimisant les approximations.

Les phénomènes étudiés par EMTP peuvent comprendre une plage de fréquences allant du courant continu jusqu'à 50 MHz [5] :

- Surtensions temporaires, résonnances, transitoires électromécaniques : 0.1 Hz à 3 kHz;
- Surtensions de manœuvres : 60 Hz à 20 kHz;
- Surtensions dues à la foudre : 10 kHz à 3 MHz;
- Surtensions dues à un réamorçage d'un arc électrique : 100 kHz à 50 MHz.

La reproduction de certains phénomènes rapides demande des données et des algorithmes mathématiques très précis. EMTP utilise aussi un processus itératif [1], [2], [3] de résolution de ses équations de réseau. De façon générale la sophistication algorithmique, les modèles détaillés et la généralisation topologique imposent des temps de calcul importants pour des réseaux contenant un grand nombre de modèles (composants) et/ou un nombre significatif de nœuds électriques.

Nous présentons ici les grandes lignes des méthodes de solution du logiciel EMTP utilisé dans le cadre du présent projet.

Les étapes standards pour effectuer une simulation dans le domaine du temps avec le logiciel EMTP sont données à la Figure 2-1 .

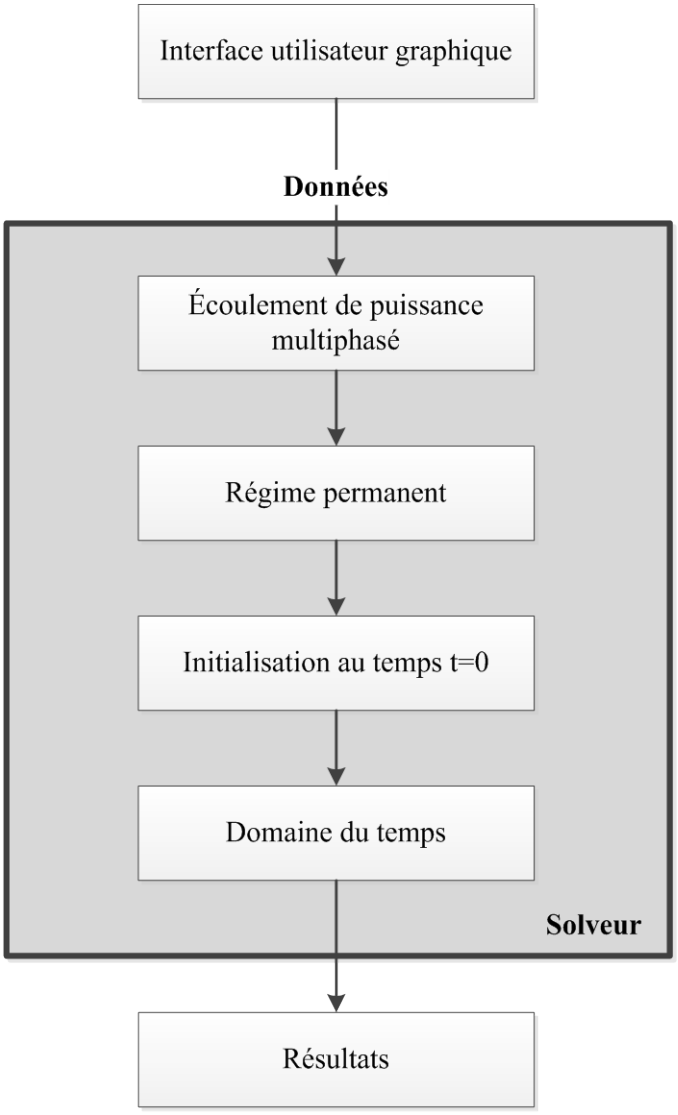

Figure 2-1 Étapes et modules de simulation du logiciel EMTP

La plupart des logiciels de simulation des circuits électriques ou électroniques utilisent l'analyse nodale classique pour la résolution des équations de circuit en régime permanent et en régime

transitoire. Les équations de l'analyse nodale, basées sur la loi de courant de Kirchhoff, sont écrites sous une forme matricielle [4] :

$$\mathbf{Y_n v_n = i_n} \tag{2.1}$$

où $\mathbf{Y_n}$ est la matrice d'admittance du réseau, $\mathbf{v_n}$ le vecteur des tensions à chaque nœud du réseau et $\mathbf{i_n}$ le vecteur de la somme des courants entrant à chaque nœud. Les caractères gras sont utilisés dans l'ensemble du document pour représenter les matrices et les vecteurs. Les admittances des composants du réseau sont combinées à chaque nœud, rendant cette méthode rapide et efficace. L'élaboration du système d'équation se fait de la manière suivante pour un élément entre un nœud *k* et un nœud *m* :

$$\begin{bmatrix} \mathbf{i_k} \\ \mathbf{i_m} \end{bmatrix} = \begin{bmatrix} \mathbf{Y_{kk}} & \mathbf{Y_{km}} \\ \mathbf{Y_{mk}} & \mathbf{Y_{mm}} \end{bmatrix} \begin{bmatrix} \mathbf{v_k} \\ \mathbf{v_m} \end{bmatrix} \tag{2.2}$$

Les vecteurs de courants $\mathbf{i_k}$ et $\mathbf{i_m}$ représentent les sommes des courants sortant des nœuds *k* et *m* respectivement. Les vecteurs de tension $\mathbf{v_k}$ et $\mathbf{v_m}$ représentent les tensions à ces nœuds. Normalement, les matrices des admittances entre nœuds $\mathbf{Y_{km}}$ et $\mathbf{Y_{mk}}$ sont égales. Par exemple, une résistance *R*, une inductance *L* ou une capacitance pure *C* entre deux nœuds *k* et *m* transmettraient les matrices d'admittance suivantes aux nœuds en question de la matrice $\mathbf{Y_n}$:

$$\mathbf{Y_R} = \begin{bmatrix} \frac{1}{R} & -\frac{1}{R} \\ -\frac{1}{R} & \frac{1}{R} \end{bmatrix} \quad \mathbf{Y_L} = \begin{bmatrix} \frac{1}{j\omega L} & -\frac{1}{j\omega L} \\ -\frac{1}{j\omega L} & \frac{1}{j\omega L} \end{bmatrix} \quad \mathbf{Y_C} = \begin{bmatrix} j\omega C & -j\omega C \\ -j\omega C & j\omega C \end{bmatrix} \tag{2.3}$$

Or, cette méthode comporte plusieurs limitations, dont l'admittance des modèles qui doit correspondre à (2.2), le fait que les sources doivent être connectées à la terre et que les interrupteurs idéaux ne peuvent être représentés [3]. C'est pourquoi le logiciel EMTP utilise plutôt l'analyse nodale *modifiée-augmentée* (MANA). Le système d'équation (2.1) est augmenté et peut être réécrit ainsi :

$$\mathbf{A_N x_N = b_N} \tag{2.4}$$

La matrice $\mathbf{A_N}$ et le vecteur $\mathbf{x_N}$ contiennent maintenant des inconnus de tension et de courant, et le vecteur $\mathbf{b_N}$ contient des valeurs connues de tension et de courant. Plus précisément, le système d'équation utilisé par EMTP est le suivant [4] :

$$\begin{bmatrix} \mathbf{Y_n} & \mathbf{A_c} \\ \mathbf{A_r} & \mathbf{A_d} \end{bmatrix} \begin{bmatrix} \mathbf{v_n} \\ \mathbf{i_x} \end{bmatrix} = \begin{bmatrix} \mathbf{i_n} \\ \mathbf{v_x} \end{bmatrix} \tag{2.5}$$

Les matrices $\mathbf{A_c}$, $\mathbf{A_r}$ et $\mathbf{A_d}$ représentent la partie augmentée des équations contenant les informations qui ne se retrouvent pas dans la matrice d'admittance $\mathbf{Y_n}$. Le vecteur $\mathbf{i_x}$ contient les inconnus de courant des modèles et $\mathbf{v_x}$, le vecteur des tensions de nœuds connues (ex : sources de tensions idéales). Les vecteurs $\mathbf{v_n}$ et $\mathbf{i_n}$ conservent la même définition qu'avec l'analyse nodale classique, soit les tensions inconnues à chaque nœuds et les courants connus respectivement. Ce système d'équations permet désormais de représenter entres autres des sources idéales entre deux nœuds, des interrupteurs et des transformateurs idéaux.

En régime permanent, l'admittance de chaque modèle est calculée selon les fréquences présentes dans le réseau puis fournie à la matrice d'admittance $\mathbf{Y_n}$. Les valeurs sont en nombres complexes puisque la solution se fait dans le domaine des phaseurs. Si le régime permanent comprend plusieurs fréquences, les vecteurs $\mathbf{x_N}$ associés à la solution de chaque fréquence sont combinés sous la forme d'une série de Fourier.

La solution dans le domaine du temps est plus complexe. La solution est calculée à chaque pas de temps et la réponse transitoire est déterminée selon des équations différentielles. Il est nécessaire de discrétiser les équations différentielles en utilisant une technique d'intégration numérique implicite pour les solutionner. Le logiciel EMTP utilise principalement l'intégration trapézoïdale, une technique répandue et dont la programmation est très efficace [5].

Une équation différentielle ordinaire s'écrit comme suit :

$$\begin{aligned} &\frac{dx}{dt} = f(x,t) \\ &x(0) = x_0 \end{aligned} \tag{2.6}$$

La solution x_t au temps t de cette équation se calcule ainsi par l'intégration trapézoïdale :

$$x_t = x_{t-\Delta t} + \frac{\Delta t}{2}\left(f_t + f_{t-\Delta t} \right) \tag{2.7}$$

Les variables $t-\Delta t$ représentent les termes historiques. La méthode d'intégration permettra de trouver la solution au temps t. Par exemple, l'équation différentielle déterminant la tension aux bornes d'une inductance pure entre les nœuds k et m est donnée par :

$$v_{km} = L\frac{di_{km}}{dt} \tag{2.8}$$

Cette relation est discrétisée par la méthode d'intégration trapézoïdale afin de calculer le courant à chaque pas de temps :

$$i_{km_t} = \frac{\Delta t}{2L} v_{km_t} + \frac{\Delta t}{2L} v_{km_{t-\Delta t}} + i_{km_{t-\Delta t}} \tag{2.9}$$

Dans l'équation précédente, la relation proportionnelle entre le courant et la tension $\frac{2L}{\Delta t}$ représente la partie résistive. Le reste de l'équation du courant dépend des valeurs de tension et de courant au pas de temps précédent, et est donc connu. On peut donc réécrire l'équation (2.9) comme étant :

$$i_{km_t} = \frac{\Delta t}{2L} v_{km_t} + i_{Lh} \tag{2.10}$$

Il est possible d'exprimer cette relation comme une résistance en parallèle avec une source de courant de valeur connue comme il est illustré à la Figure 2-2 [5].

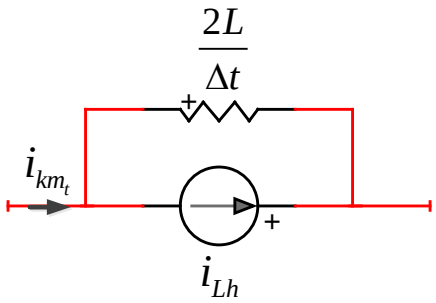

Figure 2-2 Modèle discret d'une inductance dans le domaine du temps

Puis, il suffit d'insérer ces valeurs dans la matrice d'admittance $\mathbf{Y_n}$ et dans le vecteur de courant $\mathbf{i_n}$ du système d'équations tel que décrit précédemment. À titre de second exemple, on peut exprimer la tension aux bornes d'un condensateur selon l'intégration trapézoïdale comme étant :

$$\begin{aligned} i_{km} &= C\frac{dv_{km}}{dt} \\ i_{km_t} &= \frac{2C}{\Delta t}v_{km_t} - \frac{2C}{\Delta t}v_{km_{t-\Delta t}} - i_{km_{t-\Delta t}} \end{aligned} \tag{2.11}$$

On peut donc représenter le condensateur par un modèle semblable à celui de l'inductance de la Figure 2-2. Le même raisonnement s'applique lorsqu'on utilise la méthode d'intégration d'Euler implicite (Euler Backward ou Backward Euler) à un demi-pas de temps [6]. Celle-ci est utilisée par le solveur d'EMTP lorsque le réseau est soumis à une discontinuité, par exemple, l'ouverture ou la fermeture d'un interrupteur idéal. L'alternance entre ces deux méthodes permet ainsi une meilleure précision de la simulation. La méthode d'Euler implicite propose la solution suivante pour une équation différentielle ordinaire telle que décrite à l'équation (2.6) :

$$x_t = x_{t-\Delta t/2} + \frac{\Delta t}{2} f_t \tag{2.12}$$

Avec ces méthodes d'intégration, tous les modèles contribuent au système d'équation dans le domaine du temps. Par la suite, le solveur résout à chaque pas de temps le système d'équations global qui résulte de la combinaison des modèles, avec des algorithmes de matrices creuses [7].

2.2 Modélisation des lignes de transmission

Un réseau électrique est normalement constitué de plusieurs lignes (ou câbles) de transport. Il existe plusieurs niveaux de modélisation des lignes de transport [8] :

- Modèle PI simple pour le régime permanent (50 ou 60 Hz);
- Modèle PI-exact pour les études fréquentielles;
- Modèle à paramètres constants (CP);
- Modèle à paramètres dépendants en fréquence (FD);
- Modèle de type large bande (Wideband).

Ce mémoire est basé sur l'utilisation du modèle CP, mais l'approche de parallélisation utilisée est aussi applicable aux modèles plus complexes (FD et Wideband) car ces modèles possèdent aussi un délai de propagation qui permet de découpler les réseaux sans effectuer des approximations [9].

La représentation dans le domaine du temps de la ligne de transmission par paramètres distribués est basée sur la représentation d'un élément infinitésimal de la ligne par les composants électriques de base, tel que montré à la Figure 2-3 [10] :

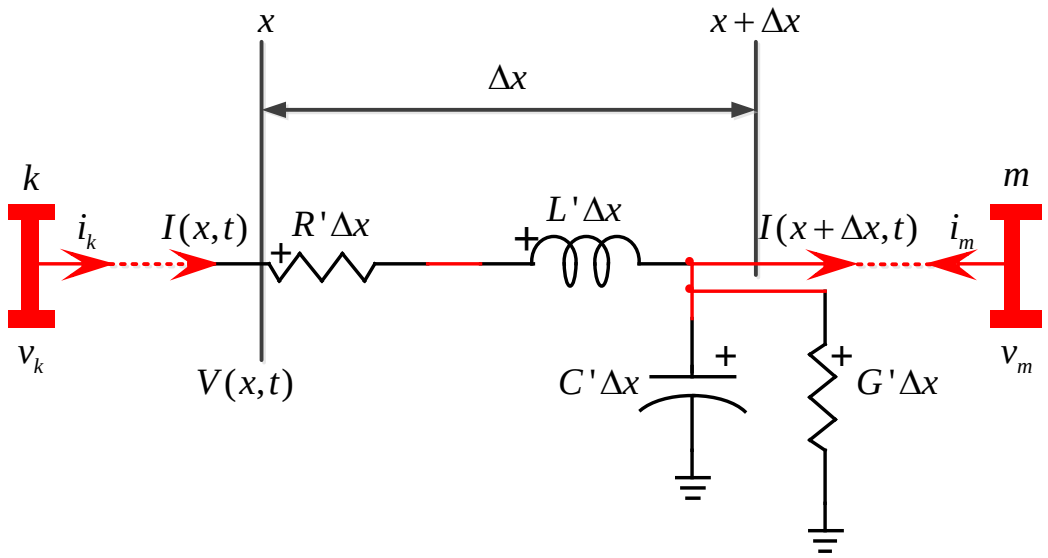

Figure 2-3 Représentation d'un élément infinitésimal d'un conducteur

Les équations dans le domaine fréquentiel d'une ligne monophasée telle qu'illustrée à la Figure 2-3 sont :

$$\frac{dV(x,t)}{dx} = -R'I(x,t) - L'\frac{dI(x,t)}{dt} \tag{2.13}$$

$$\frac{dI(x,t)}{dx} = -G'V(x,t) - C'\frac{dV(x,t)}{dt} \tag{2.14}$$

Les variables suivies du symbole *prime* (') sont données par unité de longueur. On utilise la transformée de Laplace appliquée aux équations (2.13) et (2.14) et on obtient :

$$\frac{dV(x,s)}{dx} = (R' + sL')I(x,s) = -Z'I(x,s) \tag{2.15}$$

$$\frac{dI(x,s)}{dx} = (G' + sC')V(x,s) = -Y'V(x,s) \tag{2.16}$$

En dérivant les équations (2.15) et (2.16), on obtient :

$$\frac{d^2V(x,s)}{dx^2} = \gamma^2 V(x,s) \tag{2.17}$$

$$\frac{d^2I(x,s)}{dx^2} = \gamma^2 I(x,s) \tag{2.18}$$

avec

$$\gamma = \sqrt{(R' + sL')(G' + sC')} = \alpha + j\beta \tag{2.19}$$

La variable α est la constante d'atténuation, β est la constante de phase et γ la constante de propagation. On résout les équations différentielles précédentes (2.17) et (2.18) pour obtenir :

$$V(x,s) = V^+ e^{-\gamma x} + V^- e^{\gamma x} \tag{2.20}$$

$$I(x,s) = \frac{V^+ e^{-\gamma x} - V^- e^{\gamma x}}{Z_c} \tag{2.21}$$

L'impédance caractéristique Z_c est donnée par :

$$Z_c = \sqrt{\frac{R' + sL'}{G' + sC'}} = \hat{Z}_c \angle \theta_{Z_c} \tag{2.22}$$

Afin d'obtenir le modèle en régime permanent, on obtient les équations (2.20) et (2.21) en appliquant les conditions aux frontières *k* (début de la ligne), *m* (fin de la ligne) selon la longueur de la ligne ℓ :

$$V_k - Z_c I_k = V_m + Z_c I_m \; e^{-\gamma \ell} \tag{2.23}$$

$$V_k + Z_c I_k = V_m - Z_c I_m \; e^{\gamma \ell} \tag{2.24}$$

ou

$$(V_k - Z_c I_k) e^{\gamma \ell} = V_m + Z_c I_m \tag{2.25}$$

$$(V_k + Z_c I_k)e^{-\gamma\ell} = V_m - Z_c I_m \tag{2.26}$$

On peut déduire des équations (2.23) à (2.26) et de la formulation en matrice d'admittance nodale le modèle de ligne PI-exact utilisé normalement en régime permanent par EMTP. Or, comme il sera vu à la section 3.3.4, il n'est pas possible d'utiliser cette représentation pour la ligne servant de point de coupure d'un réseau découplé. Or, la méthode alternative utilisée donne des résultats équivalents qui sont démontrés à la section 4.1 des tests de validation. Le modèle PI ne sera donc pas détaillé davantage ici.

En transformant les équations (2.23) et (2.24) vers le domaine du temps, on obtient les équations de délai de transmission pour une ligne sans pertes :

$$v_k(t) - Z_c i_k(t) = v_m(t-\tau) + Z_c i_m(t-\tau) \tag{2.27}$$

$$v_m(t) - Z_c i_m(t) = v_k(t-\tau) + Z_c i_k(t-\tau) \tag{2.28}$$

Le délai de propagation τ est défini comme :

$$\tau = \ell\sqrt{L'C'} \tag{2.29}$$

On peut réécrire les équations (2.27) et (2.28) afin de représenter le modèle temporel d'une ligne à paramètres constants en fréquence sans pertes. Le modèle est illustré à la Figure 2-4 et les termes historiques sont définis aux équations (2.29) et (2.30).

$$i_{kh} = \frac{v_m(t-\tau)}{Z_c} + i_m(t-\tau) \tag{2.30}$$

$$i_{mh} = \frac{v_k(t-\tau)}{Z_c} + i_k(t-\tau) \tag{2.31}$$

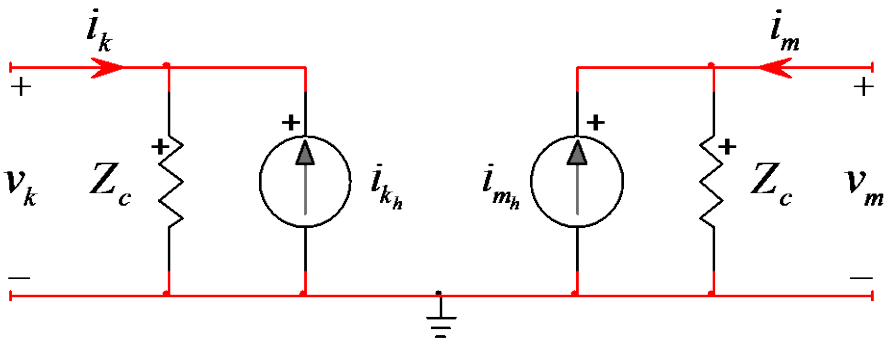

Figure 2-4 Modèle à paramètres distribués sans pertes dans le domaine du temps

Les résultats précédents ont été obtenus en négligeant les pertes résistives de la ligne. On inclut les pertes dans le modèle de ligne utilisé en distribuant la résistance totale $R = R'\ell$ de la ligne tel qu'illustré à la Figure 2-5 [11].

Figure 2-5 Inclusion des pertes dans le modèle de ligne à paramètres distribués

Comme on peut le constater, l'inclusion des pertes entraîne l'ajout d'un nœud intermédiaire au centre de la ligne de transmission. On démontre dans [11] que mathématiquement, on peut éliminer ce nœud intermédiaire tout en intégrant les pertes en remplaçant la résistance Z_c du modèle de la Figure 2-4 par :

$$Z_c' = Z_c + \frac{R'\ell}{4} \tag{2.32}$$

Comme expliqué plus haut, le modèle de la Figure 2-4 permet de découpler le réseau du côté *k* du réseau du côté *m*. Ce découplage est conservé avec l'inclusion des pertes. Ceci est illustré par l'exemple de la Figure 2-6. La matrice $\mathbf{A}$ de l'équation (2.4) pour ce réseau simple est donnée à l'équation (2.33).

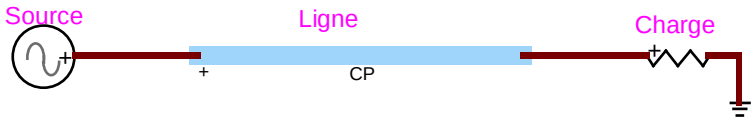

Figure 2-6 Exemple simple qui illustre le découplage

$$\mathbf{A} = \begin{bmatrix} \bullet & \bullet & \bullet & & & & \bullet & & \\ \bullet & \bullet & \bullet & & & & & \bullet & \\ \bullet & \bullet & \bullet & & & & & & \bullet \\ & & & \bullet & \bullet & \bullet & & & \\ & & & \bullet & \bullet & \bullet & & & \\ & & & \bullet & \bullet & \bullet & & & \\ \bullet & & & & & & & & \\ & \bullet & & & & & & & \\ & & \bullet & & & & & & \end{bmatrix} \tag{2.33}$$

À l'analyse de l'équation (2.33), on constate qu'il n'existe aucun lien électrique entre chaque bout de la ligne de transmission. Par conséquent, les lignes de transmission avec modélisation par paramètres distribués sont utilisées couramment dans la littérature comme point de découplage d'un grand réseau électrique (voir section 2.3) [19].

Le modèle présenté à la Figure 2-4 est applicable à une ligne monophasée. Dans le cas multiphasé, les équations différentielles du modèle CP sont décomposées en appliquant une transformation modale [8]. La transformation modale permet de solutionner les équations découplées de la ligne (modèle de la Figure 2-4) indépendamment et retourner par la suite aux équations (2.4) dans le domaine des phases en appliquant la transformation inverse.

2.3 Parallélisation des logiciels

2.3.1 Architecture parallèle des ordinateurs

Depuis plusieurs années déjà, on assiste à une augmentation constante du nombre de cœurs des processeurs des ordinateurs, sans parvenir à augmenter leur performance de manière significative. En effet, après le tournant des années 2000, on assiste à une augmentation de la cadence des processeurs de 20% en 10 ans, alors qu'on atteignait plutôt les 52% par année entre 1985 et 2000. On observe donc un plafonnement de la vitesse par processeur causé par la taille minimale des

transistors, la tension d'alimentation, la dissipation de chaleur et la vitesse limite à laquelle les électrons peuvent être transmis selon la théorie de la relativité générale [12].

La tendance est désormais à augmenter le nombre d'unités centrales de traitement (UCT, ou *CPU* en anglais) en parallèle au lieu de chercher à rendre une seule unité plus performante. Ces ordinateurs dits multi-cœurs sont l'un des types d'architectures parallèle. Il existe également les UCT en accès mémoire non-uniforme (NUMA) et les installations en « nuage », qui sont constituées d'une grappe d'ordinateurs flexibles et efficaces [13].

Bien entendu, les logiciels doivent être conçus pour exploiter au maximum le parallélisme des ordinateurs modernes. Quelques exemples des langages et méthodes de programmation les plus utilisés sont présentés à la section 2.3.3. Or, c'est le système d'exploitation qui gère l'ensemble du fonctionnement de l'ordinateur. Il sert d'interface avec les périphériques et les processus, il gère l'ensemble des types de mémoire, contrôle l'accès aux fichiers et s'occupe de l'ordonnancement des commandes et des interruptions. Il existe trois principaux systèmes d'exploitation pour ordinateur de bureau, soit *Windows*, *Mac OS X* et *Linux*. Nous mettons l'emphase sur les stratégies de parallélisation sous Windows à l'intérieur de ce mémoire étant donné que le projet de recherche a été réalisé dans cet environnement [14].

L'évaluation de la performance est centrale dans l'élaboration d'un logiciel parallèle. L'objectif principal de l'utilisation de l'architecture parallèle d'un ordinateur est de réduire le *temps réel* de l'exécution d'une tâche ou d'un programme, soit le temps tel que perçu par l'utilisateur de l'application. Il ne faut pas confondre ce temps avec le *temps d'utilisation* des différentes parties de l'ordinateur (UCT, mémoire, disque, processus), qui correspond plutôt au temps nécessité par cette ressource particulière. Ainsi, le temps réel d'exécution d'un logiciel parallèle ne correspondra pas à une fraction constante du temps d'utilisation d'un des processeurs si celui-ci n'est pas sollicité à 100% pendant toute l'exécution. L'accélération de performance est donc évaluée avec le temps réel d'exécution.

Or, la parallélisation d'un logiciel est le plus souvent appliquée à une fraction du logiciel total. En effet, ce n'est normalement pas toutes les étapes d'un processus qui peuvent être réalisées en parallèle. La loi d'Amdahl [15] permet de prédire l'accélération maximale théorique S d'un logiciel selon le nombre de processeurs p, limitée par la proportion parallélisable f :

$$S(p) = \frac{1}{(1-f) + \frac{f}{p}} \tag{2.34}$$

L'expression (1 – f) représente quant à elle la proportion non-parallélisable ou série d'un logiciel. De plus, l'efficacité *E* de la parallélisation par nombre de processeurs est évaluée par :

$$E(p) = \frac{S(p)}{p} \tag{2.35}$$

On constate par les équations (2.24) et (2.25) que lorsque le nombre de processeurs *p* augmente, l'accélération sera davantage limitée par la fraction série (1 – *f*). Cette limite théorique est plus rapidement atteinte dans la réalité par les pertes de temps qui augmentent avec le nombre de processus. En effet, ces pertes de temps, comme les surdébits de communication (*overhead*) ou les rechargements intempestifs de cache si l'affinité processeur n'est pas prise en compte, diminuent l'accélération réelle d'un logiciel parallélisé [16].

2.3.2 Mémoire partagée

La gestion de la mémoire est un enjeu crucial de la parallélisation des logiciels. Lors de l'exécution d'un programme séquentiel (sur un seul processeur), les accès à la mémoire seront faits selon une suite de lectures et d'écritures. Or, les accès en mémoire deviennent imprévisibles avec une architecture parallèle. Cette variabilité est due à l'ordonnancement. En effet, le compilateur d'un code informatique peut changer la séquence d'accès à la mémoire afin d'optimiser le programme si le résultat final reste le même. De plus, les différentes UCT ont des tampons individuels de commandes à traiter, de lecture et d'écriture, qui peuvent changer l'ordre original des tâches. Ainsi, si deux processus distincts ou deux fils d'exécution communiquent par mémoire partagée, le programmeur ne peut se fier sur l'ordre des commandes dans le code pour gérer l'accès aux données [17].

Le modèle séquentiel d'accès aux données impose que l'écriture d'une donnée soit effective partout (dans tous les espaces mémoires partagés ou non) avant d'effectuer les prochains accès en lecture ou en écriture, peu importe leur nature. Dans les faits, ce modèle est beaucoup trop contraignant pour l'exécution parallèle et réduit drastiquement la performance. La majorité des contraintes d'accès en mémoire peuvent être relâchées, et les règles à respecter doivent être

imposées par des instructions explicites de synchronisation. Ces instructions, aussi appelées *primitives de synchronisation*, s'assurent que plusieurs fils ou processus concurrents n'exécutent pas certaines *sections critiques* d'un programme en même temps selon le principe *d'exclusion mutuelle*. Ces primitives servent à éviter les situations de *concurrences critiques* aléatoires résultant de commandes exécutées dans un ordre non-prévu par le programmeur [18].

La synchronisation peut être réalisée dans une diversité de langages de programmation, souvent détaillée dans des normes. Les méthodes les plus connues appliquées à certains logiciels de simulation de réseaux électriques sont présentées dans la sous-section suivante.

2.3.3 Simulations sur architecture multi-cœurs

La simulation parallèle pour accélérer les simulations des réseaux électriques est discutée depuis plusieurs années. En effet, les différents comités internationaux anticipent, depuis plus de vingt ans, les problèmes causés par la programmation séquentielle des logiciels traditionnels couplée à la complexité grandissante des installations [19]. Après un survol des nombreuses méthodes de parallélisation de divers logiciels présentées dans la littérature, deux stratégies sont les mieux adaptées aux logiciels de simulation de réseaux.

La première consiste à paralléliser directement dans le code source les algorithmes de résolution de la matrice du réseau. Cette stratégie consiste à partager la charge de calcul sur les différents cœurs de l'ordinateur par les sections les moins couplées entres elles [20]. Ces modifications peuvent être réalisées à l'aide de l'interface de programmation pour le calcul parallèle *OpenMP*, utilisée fréquemment dans les logiciels de calcul scientifique [21]. De plus, si on désire exécuter une simulation sur une grappe de calcul, la norme MPI (dont, entres autres, l'implémentation *Open MPI*) est de loin la plus répandue dans le domaine [22]. Enfin, une autre approche consiste à utiliser une matrice de processeurs graphiques (*GPU*, en anglais) afin d'effectuer une simulation de stabilité transitoire programmée sur des réseaux tests très volumineux [23]. En ce qui a trait spécifiquement aux logiciels de résolution des circuits électriques, des travaux sont réalisés afin de paralléliser les algorithmes de résolution des matrices creuses [24], [25]. Ici, contrairement à la seconde méthode de parallélisation présenté ci-après, le réseau est modélisé à l'intérieur d'une même instance et le travail de parallélisation se fait à l'intérieur du logiciel à paralléliser.

La seconde méthode fragmente un réseau électrique en plusieurs sous-réseaux par la séparation géographique naturelle de ces derniers par des longues lignes de transport [19], comme il est mentionné à la section 2.2. Chaque sous-réseau est résolu à chaque pas de temps, puis les informations nécessaires sont transmises aux voisins. Ainsi, chaque sous-réseau est modélisé dans une instance distincte du logiciel, qui chacune s'exécute sur une unité de calcul différente. De plus, ceci rend possible l'utilisation de pas de temps différents dans chaque instance, permettant ainsi d'isoler des événements très rapides survenant dans un sous-réseau et de résoudre les autres à des pas de temps plus grands, donc avec moins de calculs [26]. Aussi, cette méthode permet d'obtenir des matrices plus petites, qui sont moins longues à refactoriser lors des itérations faites par le solveur lors d'une non-linéarité (ex. défaut). Cette méthode est utilisée en pratique depuis l'avènement des logiciels de simulation des réseaux électriques en temps réel comme RT-LAB, Hypersim et RTDS [27], [28], [29], [30]. Très récemment, des travaux ont également été réalisés sur d'autres logiciels de calculs transitoires électromagnétiques qu'EMTP, utilisant cette méthode de découplage d'un réseau de base. En utilisant des cas de tests théoriques très volumineux, des gains de performance sont observés. Un des groupes de recherche utilise une interface existante de communication (ENI) basée sur le protocole TCP/IP [31], tandis que le second utilise des canaux nommées (*named pipes*) pour l'échange de données entre les instances [32].

Une stratégie de communication employée pour implémenter la seconde méthode (découplage), moins présente dans la littérature pour le type d'application discuté ici, est le *sémaphore* [33]. Le sémaphore est une primitive de synchronisation de bas niveau, utilisé autant comme verrou que comme variable de condition afin de protéger les opérations de lecture et d'écriture sur des données partagées. Le sémaphore donne la possibilité que plusieurs processus entrent en même temps dans une *zone critique* de code avant de bloquer l'accès. Or, la majorité des applications pratiques (dont celle qui fait l'objet de ce mémoire) permettent uniquement l'exclusion mutuelle simple, donc qu'un seul fil d'exécution ou processus accède à la zone critique à la fois. Dans la plupart des cas, on parlera de *mutex*. Les utilisateurs du système d'exploitation Windows peuvent facilement utiliser les objets sémaphores [34] en utilisant trois fonctions simples, disponibles dans l'en-tête *windows.h*. Tout d'abord, la fonction *CreateSemaphore* créée une variable sémaphore et détermine les décomptes initial et maximal, soit le nombre de processus pouvant accéder à la zone critique. Lorsqu'un processus désire accéder à la zone critique, il utilisera la

fonction *WaitForSingleObject* afin de savoir s'il est permis d'y accéder selon le décompte actuel de la variable. Si le décompte du sémaphore était à 0, le processus attendra que le sémaphore se libère si une valeur de temporisation a été spécifiée. Si le décompte était supérieur à 0, le sémaphore décrémentera le décompte de 1. Finalement, lorsqu'un processus a terminé d'accéder à la zone critique, il libère la zone critique en appelant la fonction *ReleaseSemaphore*, qui incrémente le décompte du sémaphore de 1.

Une autre méthode de synchronisation de bas niveau est le verrouillage total de l'UCT (ou *spinlock*, en anglais). Contrairement au sémaphore, ce mécanisme d'exclusion mutuelle contraint le processus en attente à rester actif tout en monopolisant l'UCT, d'où le terme « verrouillage total » [36]. Cette méthode est utilisée dans les applications où un démarrage rapide d'une tâche est requis, par exemple dans les applications temps réel.

Finalement, une quantité impressionnante de normes pour plusieurs domaines d'application existent afin de baliser les communications entre des applications exécutées en parallèle. L'outil présenté à l'intérieur de ce mémoire est basé sur la norme FMI [35] pour la co-simulation, norme utilisée pour le couplage de deux ou plusieurs outils de simulation selon une méthode de communication maître-esclaves. La norme couvre les applications de co-simulation entre deux types de logiciels (par exemple, les parties « puissance » et « contrôle » d'une simulation dans deux logiciels différents) mais également entre deux mêmes logiciels, ce qui est le cas ici.

Plusieurs stratégies d'implémentation de la parallélisation applicables au logiciel EMTP ont été présentées. La stratégie retenue est l'utilisation de sémaphores pour le partage de données et le modèle de communication est basé sur la norme FMI. Ces choix sont expliqués et détaillés au Chapitre 3, qui traite de l'implémentation de la solution proposée.

CHAPITRE 3 IMPLÉMENTATION DE LA SOLUTION PROPOSÉE

Dans ce chapitre, le choix de la solution retenue est expliqué par rapport aux méthodes présentées dans la revue de littérature. Puis, le fonctionnement de l'outil de base est détaillé. Ensuite, une explication est fournie sur les modifications, ajouts et divers choix de design requis pour réaliser un outil fonctionnel et performant.

3.1 Stratégie de parallélisation retenue

Parmi les stratégies de parallélisation présentées au chapitre précédent, c'est la méthode par découplage des réseaux de simulation qui a été retenue. Dans cette approche, un réseau donné sera séparé en plusieurs sous-réseaux et solutionné par plusieurs instances du logiciel EMTP sur le même ordinateur et en utilisant plusieurs cœurs de calcul.

La méthode par découplage a aussi été favorisée puisqu'aucune modification dans le code source du solveur n'est requise et une accélération considérable peut être observée, en autant que le moyen de communication soit efficace. Cette méthode parallélise les calculs de manière externe au logiciel en utilisant plusieurs instances de ce dernier de manière concurrente. Il est par contre requis de développer un code indépendant pour le modèle de ligne CP pour permettre le découpage multi-cœurs.

Ainsi, le moyen de communication employé est le second choix méthodologique de base du projet de recherche après la stratégie de parallélisation. Comme il a été vu au chapitre précédent, plusieurs méthodes existent et chaque projet de parallélisation en emploie une différente. Pour le présent projet, l'utilisation des sémaphores a été préférée, pour plusieurs raisons. Tout d'abord, l'utilisation de primitives de bas niveau, contrairement à une interface plus complexe, rend la communication plus simple et donc plus efficace. De plus, un outil de co-simulation basé sur EMTP existait avant le démarrage des travaux de ce mémoire et pouvait être récupérer et améliorer dans des délais raisonnables. Cet outil de co-simulation est décrit à la section 3.2.

Finalement, il faut mentionner que l'utilisation du sémaphore a été préférée au verrouillage total de l'UCT (ou *spinlock*, en anglais). En effet, comme mentionné à la section 2.3.3, l'avantage d'utiliser des sémaphores ici est que le processus en attente sera mis en veille et l'unité de calcul pourra travailler à autre chose. En pratique, en utilisant l'outil proposé avec un ordinateur

personnel pour exécuter une simulation parallèle sur l'ensemble de ses cœurs, l'utilisateur pourra continuer à utiliser certains logiciels simples et l'ordinateur ne sera ainsi pas monopolisé.

3.2 Outil de base de co-simulation

3.2.1 Objectifs de l'outil

En 2012, un ingénieur d'Électricité de France (EDF), M. Xavier Legrand, a réalisé un outil permettant de partager des signaux de contrôle d'une simulation EMTP avec le logiciel MATLAB ou avec une autre instance du logiciel EMTP. L'application première de cet outil était de pouvoir utiliser, dans une simulation EMTP, des systèmes de contrôle modélisés sous MALTAB-Simulink. En effet, les entreprises qui conçoivent les systèmes de contrôle des équipements électriques doivent habituellement aussi fournir le modèle informatique pour la simulation, mais s'assurent de le coder à l'intérieur d'une boîte noire afin que les secrets industriels du produit soient préservés. Or, plusieurs entreprises fournissent le code dans un fichier MATLAB-Simulink, alors que les simulations de puissance des gestionnaires de réseau se font habituellement sous d'autres logiciels, dont EMTP. Il y avait donc un besoin de pouvoir simuler le contrôle de l'équipement électrique tout en simulant le comportement adéquat du réseau électrique. La solution trouvée à cette problématique est la co-simulation. Les explications ainsi que les exemples présentés aux sections 3.2.1 et 3.2.3 sont tirées de [37].

Cet outil de base de co-simulation s'utilise de la manière suivante : prenons l'exemple suivant d'une machine avec régulateurs de vitesse et de tension connectée à un réseau, présenté à la Figure 3-1. Les signaux de contrôle d'entrée et de sortie de la machine synchrone *SM2* sont connectés aux blocs *HTG* et *ExcSystem* modélisant la régulation de vitesse et la régulation de tension, respectivement. Or, dans l'éventualité où les blocs de régulation ne seraient pas disponibles sous EMTP, l'utilisateur pourrait insérer l'outil de co-simulation dans une simulation EMTP tel que fait à la Figure 3-2.

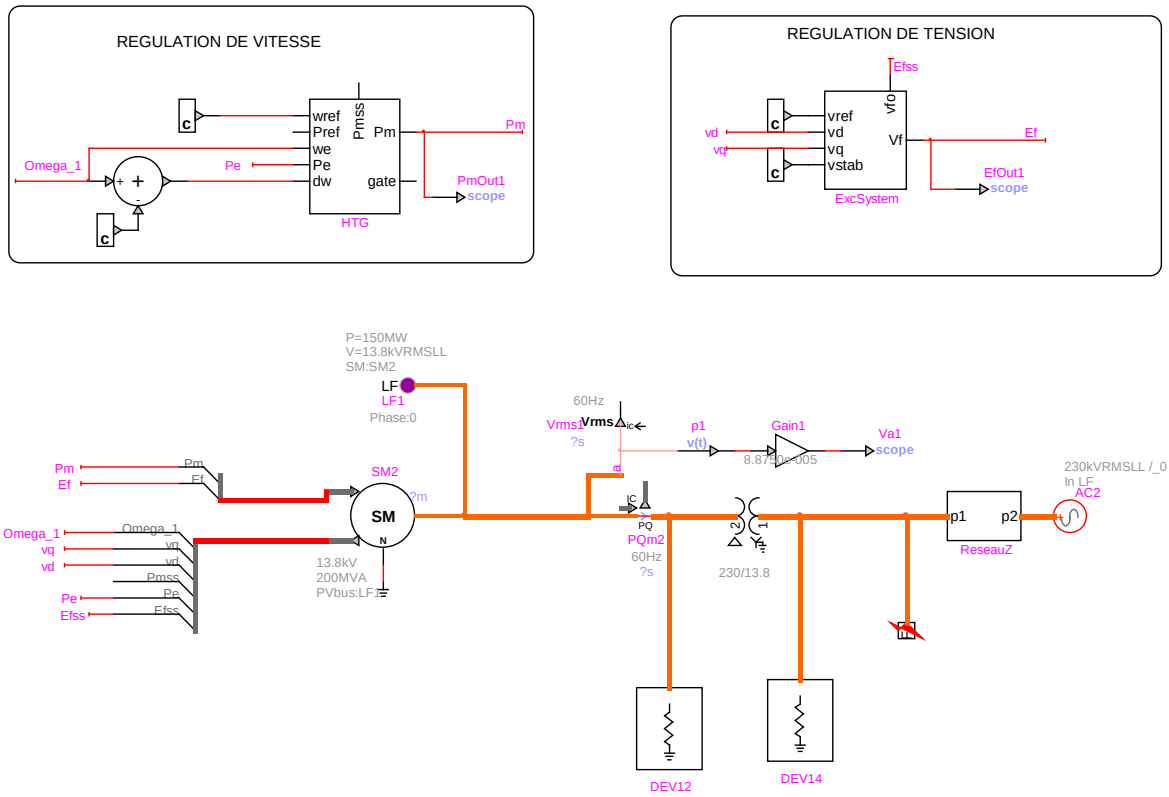

Figure 3-1 Outil de base, exemple sans co-simulation

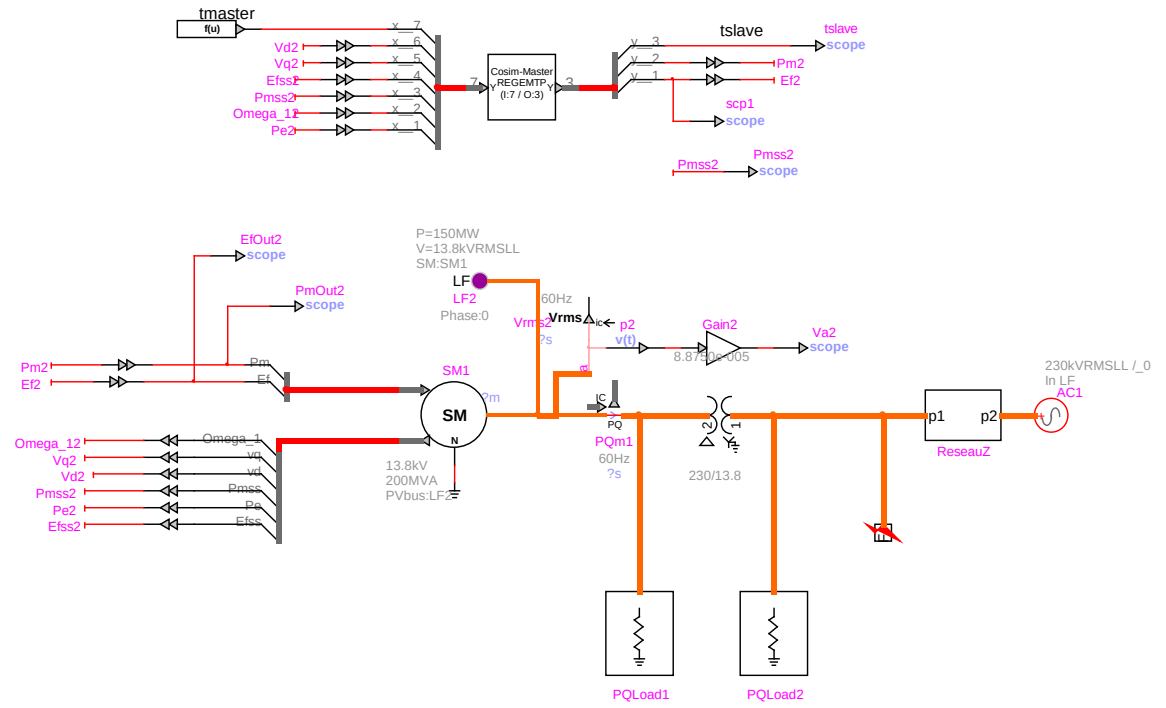

Figure 3-2 Outil de base, exemple avec co-simulation, système de puissance

Avec l'outil de base de co-simulation, tous les signaux de contrôle de la machine synchrone transitent maintenant par un bloc placé par l'utilisateur appelé *Cosim-Master*, car l'instance simulant les signaux de puissance est l'instance « maître » selon la norme FMI. Les entrées, à gauche, représentent les signaux que l'on désire envoyer au logiciel servant à modéliser le système de contrôle, soit MATLAB-Simulink ou une autre instance d'EMTP. Les signaux sortants, à droite du bloc, représentent les données qui ont été calculées par ledit logiciel et qui doivent maintenant asservir le comportement de la machine synchrone au prochain pas de temps. Du côté du système de contrôle (instance « esclave », selon FMI), un bloc de co-simulation est placé où les entrées correspondent aux sorties du bloc du côté puissance, et vice-versa. Un exemple de ceci, modélisé sous EMTP, est montré à la Figure 3-3.

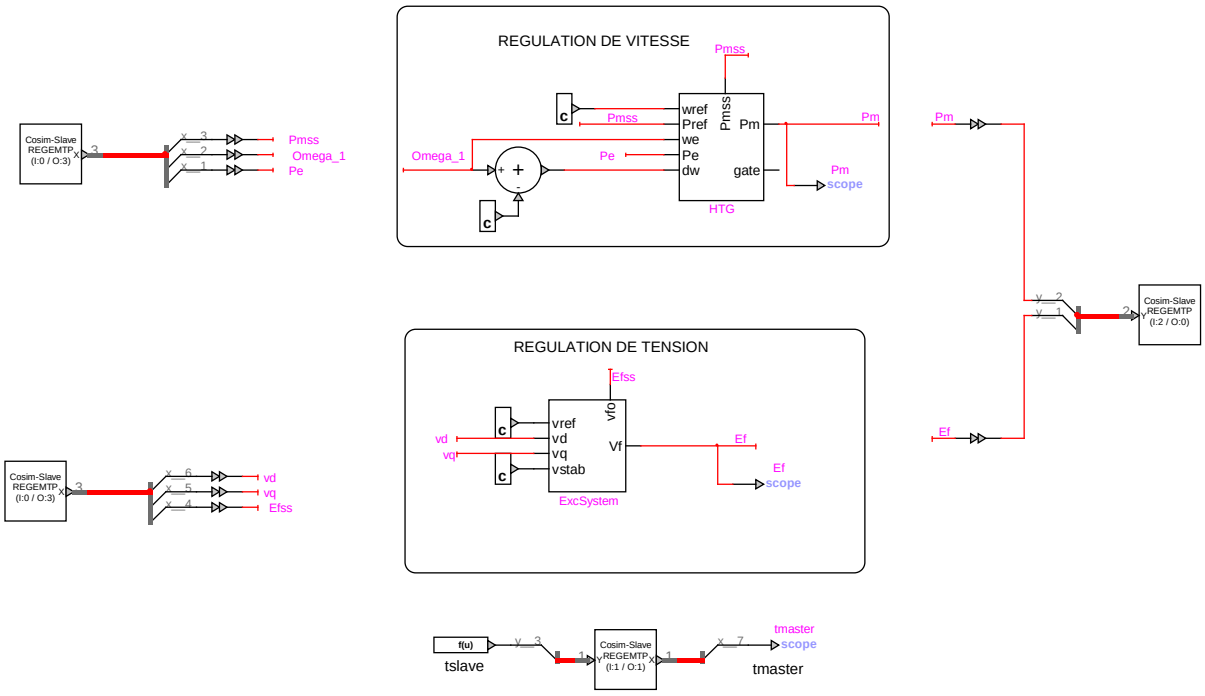

Figure 3-3 Outil de base, exemple avec co-simulation, système de contrôle

Trois modes de co-simulation ont été implémentés dans l'outil de base, correspondant à trois séquences distinctes des appels aux sémaphores. Le mode 1 est le plus stable et le plus simple mais il ne permet pas que plusieurs instances esclaves s'exécutent en même temps, ni que la

grandeur des pas de temps soit différente entre les instances. Le mode 2, quant à lui, emploie un ordre de communication qui permet les calculs en parallèle sur plusieurs instances esclaves. Finalement, le mode 3 fonctionne de manière à ce que le pas de temps du maître puisse être différent de celui de l'esclave. Les schémas de principe des trois séquences d'appels possibles sont disponibles à la référence [37].

L'outil est implémenté à l'aide d'une fonctionnalité du logiciel EMTP qui est l'ajout de modèles par les utilisateurs à l'aide de blocs DLL. La section 3.2.2 explique le fonctionnement des blocs DLL dans EMTP, puis les explications relatives aux détails d'implémentation de l'outil de base de co-simulation sont données à la section 3.2.3.

3.2.2 Fonctionnement des objets DLL dans EMTP

La fonction DLL du logiciel EMTP permet aux utilisateurs de développer des modèles avancés de systèmes de contrôle ou d'équipements de puissance et de les connecter directement à l'intérieur d'un schéma-bloc d'un réseau EMTP. Le modèle peut être écrit dans plusieurs langages informatiques avec un protocole de type API prescrit dans la documentation d'EMTP [38]. Afin de connecter un code-usager à EMTP, l'usager doit tout d'abord créer un bloc « boîte noire » vide dans l'interface graphique. Puis, tout dispositif placé sur le schéma possède plusieurs attributs ayant chacun une fonction distincte. Le programmeur doit entrer des informations obligatoires dans trois attributs afin que le programme soit fonctionnel :

- *ModelData* : Nom ou chemin d'accès complet du DLL;
- *Part name* : « DLL », afin que le solveur traite ce dispositif comme tel;
- *ParamsA* : Informations relatives au modèle comme tel.

Les informations à entrer dans *ParamsA* font référence au nombre de signaux de puissance et de contrôle, au nombre de ligne à ajouter pour la partie augmentée de la matrice $\mathbf{A}$ (par exemple, trois lignes pour la présence d'un interrupteur triphasé) et au nombre de lignes à ajouter au vecteur $\mathbf{b}$ pour les sources de courant. Ces informations sont entrées au préalable par le programmeur et le modèle est fonctionnel à l'exécution de la simulation. Aucune procédure supplémentaire de lien (*linking*) n'a besoin d'être suivie et ces informations sont gardées en mémoire dans le bloc DLL, ce qui rend les modèles utilisateurs DLL portables et flexibles.

L’objectif du fichier DLL codé par l’utilisateur est de répondre, par l’ensemble de sa bibliothèque de procédures, aux différentes requêtes envoyées par le solveur d’EMTP. Les différentes requêtes du solveur visent à remplir le système d’équations nodales modifiées-augmentées, comme mentionné à la section 2.1. Ainsi, ce dernier envoie plusieurs requêtes à chaque appareil selon le moment de la simulation. Les requêtes principales, pour les équipements de puissance et de contrôle, sont données à la Figure 3-4.

Ainsi, toutes les requêtes (encadrés pleins) de la Figure 3-4 correspondent à une procédure qui doit être remplie par l’utilisateur d’EMTP désirant programmer un modèle avancé DLL. Le programme appelant ces procédures est le solveur d’EMTP. Celui-ci envoie certaines informations (ex. la fréquence ou l’index des lignes concernées) de manière prédéterminée comme arguments à ces fonctions. Les actions du solveur, soit la résolution des systèmes de contrôle et de puissance une fois que tous les dispositifs du schéma y ont contribué, sont indiquées dans les encadrés pointillés.

À l’aide de certains appels de fonctions définies dans une des bibliothèques fournies avec le logiciel, le programme a accès, tout au long de la simulation, à plusieurs valeurs comme le temps *t* de la simulation, les valeurs actuelles dans les matrices, etc. Plusieurs autres bibliothèques sont disponibles pour assister le programmeur et surtout s’assurer que le programme respecte certaines normes du logiciel.

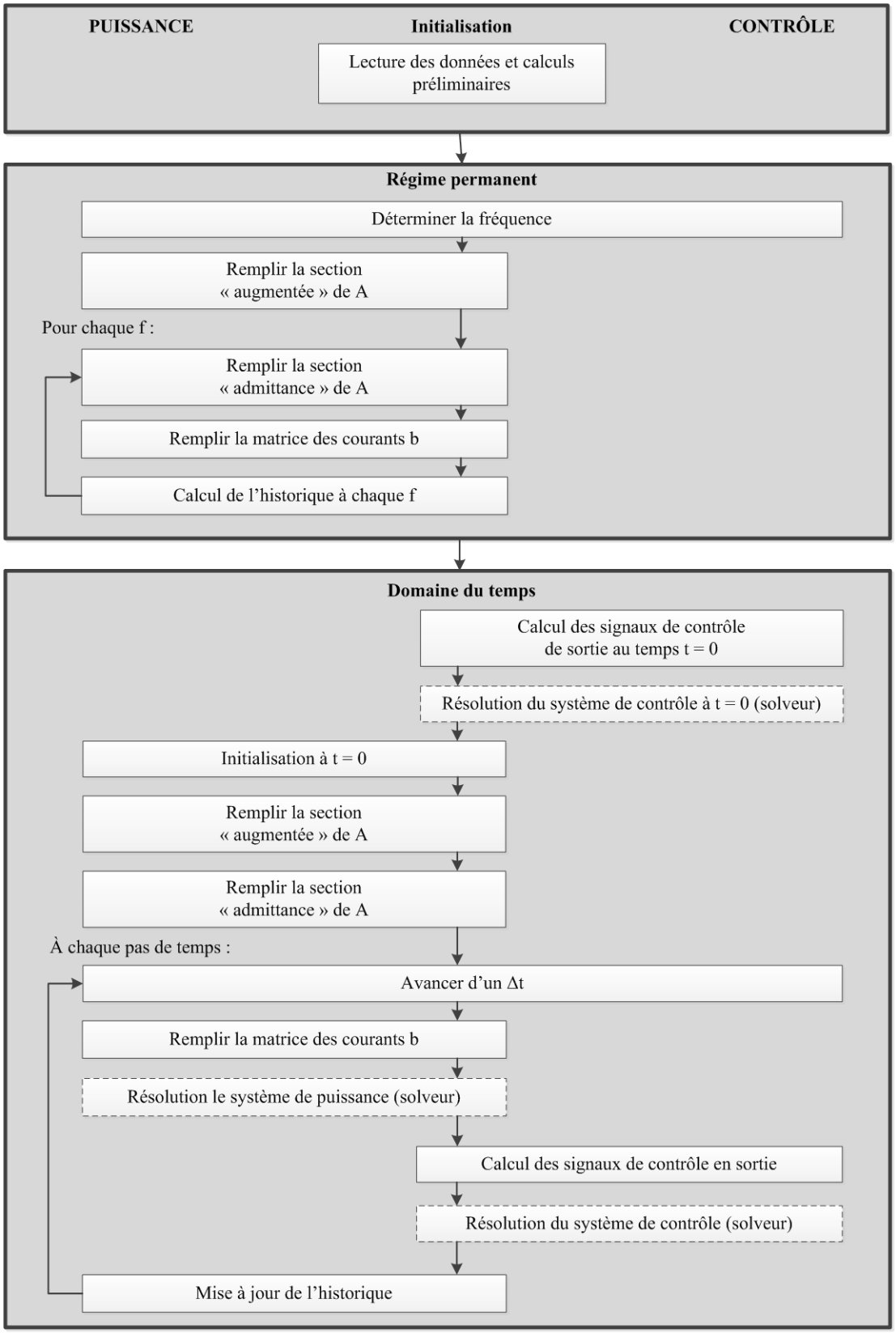

Figure 3-4 Ordonnancement des requêtes du solveur adressées aux éléments du schéma-bloc

3.2.3 Détails d'implémentation

Il a été vu à la section 3.2.1 que l'outil de base sert à lier des signaux de contrôle d'une instance maître vers une instance esclave (EMTP ou MATLAB-Simulink). Puis, il est mentionné à la section 3.2.2 qu'un utilisateur d'EMTP peut se servir des blocs DLL pour programmer des modèles simplement en répondant aux requêtes prédéterminées du solveur. La présente section détaille l'implémentation de l'outil en utilisant cette fonctionnalité d'EMTP.

Comme il a été mentionné aux sections 2.3.3 et 3.2.1, la norme FMI balise les communications entre deux instances selon un schéma de communication maître-esclave. Le rôle des instances « maître » et « esclave » étant différent, les procédures associées à celles-ci sont différentes. De plus, la structure du code est faite de manière à réaliser deux bibliothèques de routines distinctes; la première pour interfacer directement avec le solveur d'EMTP, et la seconde contenant les fonctions de gestion de la mémoire dont les appels aux sémaphores. Ces deux concepts combinés, on obtient quatre fichiers sources différents, dont les appels sont ordonnancés tel qu'indiqué à la Figure 3-5.

Les deux DLLs *Cosim-maître* et *FMI-maître* sont, au niveau de la structure du code, le miroir des DLLs *Cosim-esclave* et *FMI-esclave*. La structure des fichiers DLL *Cosim-maître* et *Cosim-esclave* correspond à la structure des modèles DLL devant se coupler avec EMTP. Les procédures des DLLs *FMI-maître* et *FMI-esclave* correspondent à ce qui est prescrit dans la norme FMI. Le rôle des deux DLLs *Cosim-X* est donc d'appeler les procédures définies dans les DLLs *FMI-X* selon les requêtes du solveur et selon le schéma de principe décrit à la Figure 3-7. Le détail des moments d'appel des différentes fonctions FMI de partage des données est donné à la Figure 3-6. À titre d'exemple, le mode 2 est représenté dans cette figure, mais l'emplacement des appels aux fonctions FMI et le contenu de celles-ci restent toujours le même peu importe le mode de communication.

Les appels aux trois objets sémaphores de Windows sont également représentés à la Figure 3-6 par les flèches et décrits par le texte en italique. On voit ainsi qu'une combinaison de trois sémaphores par bus de co-simulation est utilisée pour assurer un ordonnancement adéquat entre l'instance maître et l'instance esclave. En effet, au moment de l'initialisation, l'esclave se connecte au bus de co-simulation une fois que le maître a libéré le sémaphore #1. Puis, l'esclave rend la main au maître en libérant le sémaphore #2 pour que les deux puissent terminer

l'initialisation et commencer la simulation dans le domaine du temps. Au cours d'un pas de temps de la simulation, la même méthode de communication est répétée, cette fois avec les sémaphores #3 et #2 au lieu de #1 et #2 respectivement. C'est l'instance maître qui s'occupe de la réinitialisation des sémaphores au début et à la fin de la simulation.

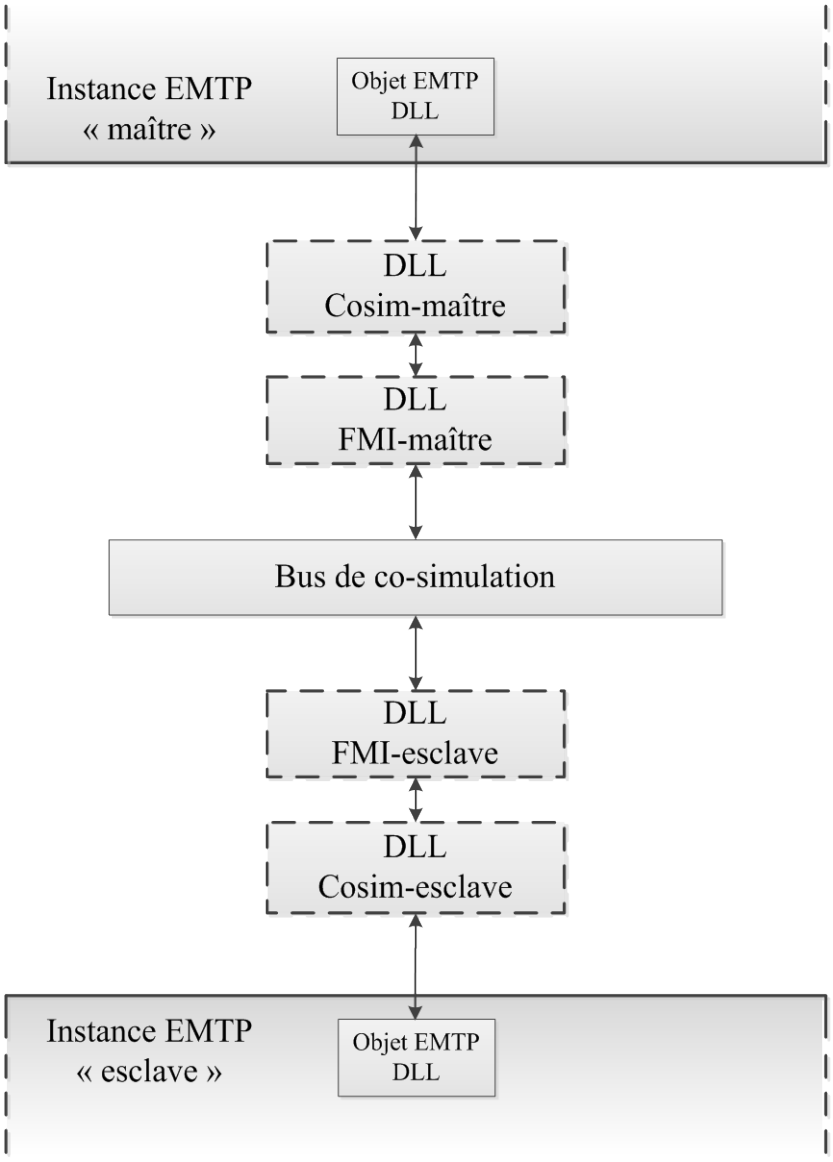

Figure 3-5 Schéma de communication des fichiers de code DLL

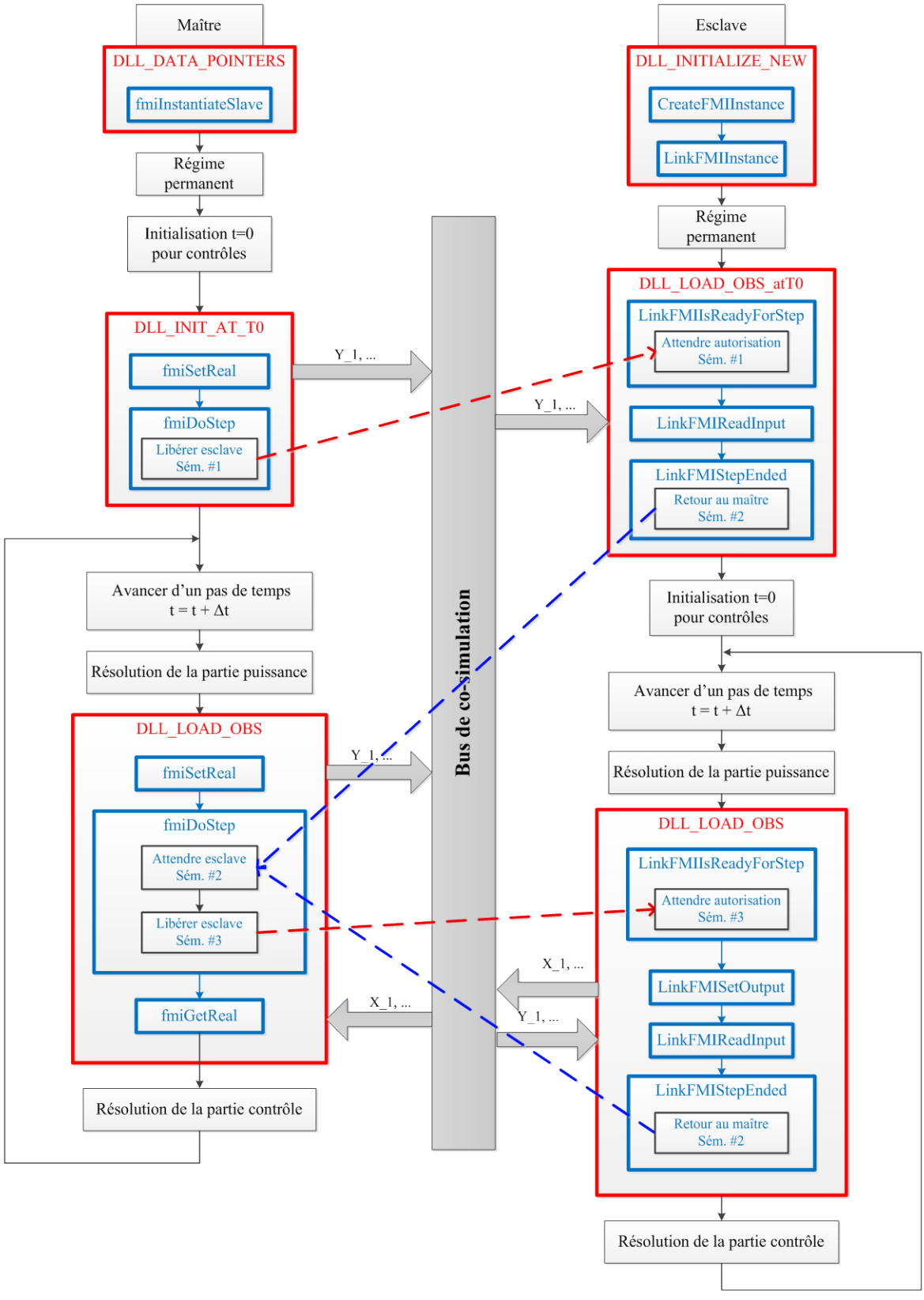

Figure 3-6 Appels des fonctions sémaphores, outil de co-simulation

La description de chaque fonction *FMI-maître* et *FMI-esclave*, selon [35] et [37], est donnée ici.

FMI-maître

- À l'initialisation :
    - *fmiInstantiateSlave* : Création du bus de co-simulation et attente de la connexion d'un esclave.
- À chaque pas de temps :
    - *fmiGetReal* : Lecture du bus de co-simulation.
    - *fmiSetReal* : Écriture dans le bus de co-simulation.
    - *fmiDoStep* : Le maître donne l'autorisation à l'esclave d'avancer d'un pas de temps en libérant le sémaphore, puis le récupérera à la fin de celui-ci.
- À la fin de la simulation :
    - *fmiTerminate* : Demande à l'esclave d'arrêter sa simulation et réinitialise les trois sémaphores.

FMI-esclave

- À l'initialisation :
    - *CreateFMIInstance* : Créer une instance esclave et donner le nom du bus.
    - *LinkFMIInstance* : Vérifier qu'un maître attend l'instance selon le nom défini par la fonction *CreateFMIInstance*, et se connecter au bus.
    - *LinkFMIPGeneration-RAZIOFMIFile,-AddIOFMIFile, -WriteFMIFile* : Initialiser le fichier descripteur .fmi du bus, y ajouter le nombre d'entrées/sorties et créer le fichier en conséquence (qui contient le nombre d'entrées et de sorties).
- À chaque pas de temps :
    - *LinkFMIIsReadyForStep* : L'esclave se met en attente de l'autorisation du maître d'avancer au prochain pas de temps en réservant le sémaphore lorsque ce dernier est disponible.
    - *LinkFMISetOutput* : Écrire les sorties dans le bus de co-simulation.

- *LinkFMIReadOutput* : Lire les entrées du bus de co-simulation.
- *LinkFMIStepEnded* : Appelé à la fin du calcul du pas de temps actuel pour le bloc DLL, signale au maître la fin du pas de temps en libérant le sémaphore.

- À la fin de la simulation :
  - *fmiTerminate* : Accomplir les actions nécessaires à l'arrêt de la simulation, dont la libération du sémaphore #2.

Finalement, il est mentionné à plusieurs reprises que le bus de co-simulation est le lien de communication entre les instances qui se partagent de l'information. Plus précisément, le bus de co-simulation est implémenté par un fichier mappé en mémoire (MMF ou *memory-mapped file* en anglais). Un fichier mappé en mémoire est un fichier sur disque directement assigné, octet par octet, avec une section de mémoire partagée entre l'instance maître et esclave de la co-simulation. Cette corrélation entre le fichier et l'espace mémoire virtuel fait en sorte que les applications utilisent cet espace comme de la mémoire vive (RAM). On note trois avantages à employer cette méthode. Tout d'abord, la mémoire est accédée directement par l'UCT, ce qui augmente la vitesse des activités d'entrées/sorties (I/O). De plus, cette méthode permet à deux applications de « mapper » le même fichier, nonobstant les multiples protections des systèmes d'exploitation modernes. L'accès aux fichiers partagés, ce qui est exactement le cas ici, est donc plus rapide. Aussi, puisque les écritures sur disque sont différées, le nombre total d'opérations d'entrées/sorties est réduit et l'exécution est plus efficace.

La création du fichier mappé en mémoire incombe à l'instance maître de la co-simulation. Cette tâche est réalisée par la fonction *FMIInstantiateSlave*, décrite dans le fichier de code *FMI-maître*. Pour ce faire, le maître utilise deux fonctions faisant partie des fonctions de base de Windows, *CreateFileMapping* et *MapViewOfFile* [39]. La première crée l'objet mappé en mémoire d'une grandeur fixe selon le nombre d'entrées/sorties déterminées par les données fournies par l'utilisateur. Un accès en lecture et écriture est accordé à tous les processus qui auront mappé ce fichier. La seconde fonction, *MapViewOfFile*, mappe cet objet dans une adresse, ici une variable faisant partie de la classe FMI du maître décrite selon [35]. Il faut préciser que le maître instancie un MMF par bus de co-simulation, soit pour chaque ligne de transmission entre un maître et un esclave dans un contexte de parallélisation. Le nom du bus de co-simulation, unique pour chaque

lien entre le maître et l'esclave, est connu par ces instances selon les données entrées par l'utilisateur et est requis au moment du mappage.

Ce fichier/bus de co-simulation est également rendu visible à l'instance esclave par la fonction *LinkFMIInstance*, qui appelle elle aussi la fonction *MapViewOfFile* après le maître. Ainsi, le fichier mappé en mémoire est accessible par les instances maître et esclave à chaque extrémité d'un bus de co-simulation. Les accès en lecture/écriture à ce fichier par les instances sont synchronisés par les sémaphores selon le schéma de la Figure 3-6. La mécanique de création et d'accès au fichier mappé expliquée ici est conservée dans le nouvel outil de parallélisation développé dans le cadre du projet et décrit à la section 3.3, même si l'ordre d'appel des sémaphores est alors modifié.

Ainsi, le fichier mappé en mémoire est utilisé comme un tampon de données qui sert à partager plusieurs informations. La nature et l'emplacement de chaque variable à être partagée sont normalisés selon [35], et sont respectés par l'outil de base et par l'outil de parallélisation qui est présenté plus loin. La description du bus de co-simulation est donnée au Tableau 3-1.

Tableau 3-1 Description du bus de co-simulation

Index	**Description de la variable**
0	Nombre d'entrées (vers esclave)
1	Nombre de sorties (vers maître)
2	Mode de co-simulation (1, 2 ou 3)
3	Temps *t* esclave (synchronisation)
4	Δt esclave
5	Temps *t* maître (synchronisation)
6	Δt maître
7	Signal d'erreur maître
8	Signal d'erreur esclave
9	*Y_1*
…	…
8 + n entrées	*Y_NbEntrées*
9 + n entrées	*X_1*
…	…
8 + n (entrées + sorties)	*X_NbSorties*

3.3 Développement de l'outil et optimisation pour la parallélisation des simulations

À la section 3.2, l'utilisation des blocs DLL dans EMTP et l'implémentation d'un *outil de co-simulation* respectant la norme FMI ont été présentées pour le partage de signaux de contrôle. Le corps de ce projet de recherche fut d'apporter plusieurs ajouts et modifications à partir de l'outil de base de façon à créer un nouvel *outil de parallélisation* du logiciel. C'est l'ensemble de ces modifications et de ces ajouts qui font l'objet de la présente section.

3.3.1 Intégration des signaux de puissance

Une modification cruciale est le fait de devoir partager des signaux de puissance au lieu des signaux de contrôle. Ici, il ne suffit pas de changer la nature des ports de connexion du bloc. En effet, comme il a été vu à la section 3.2.2 et plus précisément à la Figure 3-4, les requêtes du solveur au bloc DLL ne sont pas les mêmes selon que le bloc possède des ports de puissance et/ou des ports de contrôle. Ces informations sont indiquées dans le champ *ParamsA* des attributs. Il a donc fallu refaire l'ensemble du schéma de communication afin d'introduire les appels sémaphores dans les requêtes liées au système de puissance, tout en gardant un séquencement des communications stable et fonctionnel. La Figure 3-7 montre ce nouveau schéma d'ordonnancement des appels sémaphores à l'intérieur des requêtes du système de puissance. C'est encore une fois le mode 2 qui est montré ici, puisque c'est celui qui est utilisé pour l'ensemble des résultats de performance montrés au Chapitre 4. En effet, des trois modes de communications énumérés à la section 3.2.1, c'est le seul qui permette de réaliser des calculs en parallèle pour chaque esclave. C'est donc avec ce mode que l'outil de parallélisation a été testé afin de maximiser la rapidité de la simulation, même si l'infrastructure des deux autres modes a été laissée en place pour d'éventuels besoins futurs.

De plus, les entrées/sorties du bus de co-simulation (Tableau 3-1) ont été fixées à 3 ou 6 « entrées » (de l'instance maître vers l'instance esclave), et le même nombre vers l'instance maître. Le nombre de variables partagées varie selon le type de ligne (standard ou biterne).

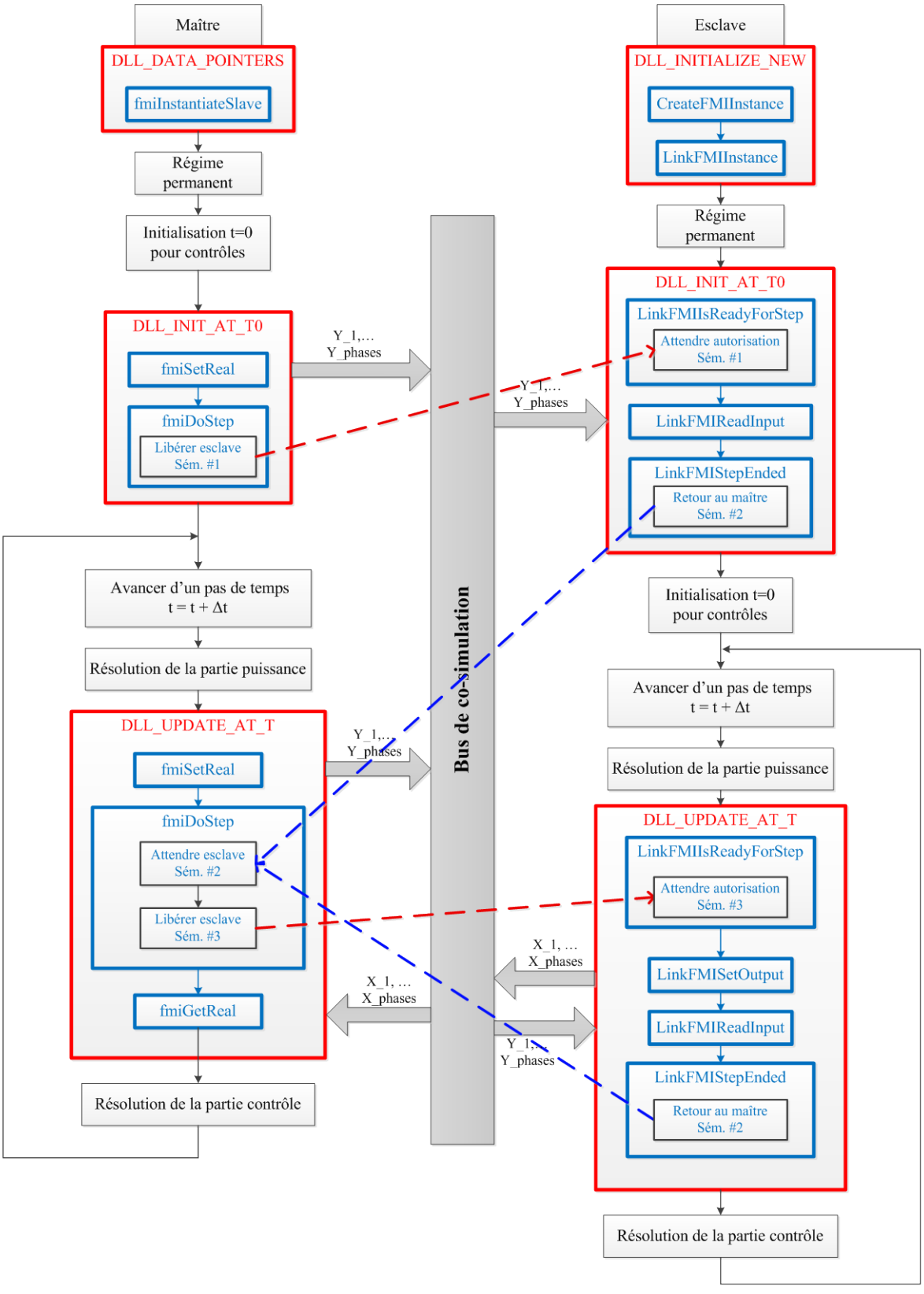

Figure 3-7 Appels des fonctions sémaphores, nouvel outil de parallélisation

Un autre aspect du design de l'outil original a dû être modifié, avec l'intégration des signaux de puissance, en ce qui a trait aux types d'intégration possibles, soit trapézoïdale ou Euler implicite. Vous pouvez vous référer à la section 2.1 pour les détails de ces deux méthodes. En effet, l'outil original avait été conçu de manière à transiter l'information aux pas de temps entiers seulement, et non pas aux demi-pas de temps lorsque le solveur passe à la méthode d'Euler implicite. Ainsi, à l'intérieur de la procédure qui répond à la requête *req_load_observables* du solveur (voir Figure 3-4) du fichier DLL *Cosim-maître*, le développeur s'était assuré que les appels sémaphores n'étaient faits que lors des pas de temps entier. Pour ce faire, il s'était servi d'une variable mise à la disposition de tous les utilisateurs d'EMTP désirant développer des modèles DLL. Cette variable, *OnTimeMesh*, est disponible à partir du module Fortran *simulation_data* fourni avec le logiciel. Ceci est une variable booléenne, mise à jour par le solveur à chaque pas de temps, qui permet de savoir si le pas de temps actuel est un nombre entier (1) ou sur un demi-pas de temps (0). Or, ce changement de type d'intégration se fait seulement si la partie « puissance » de la situation le requiert. Ce changement ne se produit pas dans l'instance esclave, que ce soit une instance EMTP ou MATLAB-Simulink. Ainsi, il était crucial, dans l'outil de co-simulation original, que le partage de données se fasse uniquement lors les pas de temps entier. Ceci était réalisé simplement en incluant tous les appels sémaphores de la procédure *req_load_observables* du DLL *Cosim-maître* dans une boucle *if* qui vérifie l'état de la variable *OnTimeMesh*. Or, en intégrant les signaux de puissance, il était possible que le solveur de l'esclave change le mode d'intégration indépendamment du solveur de l'instance maître, et vice-versa. C'est pourquoi il a aussi fallu intégrer les appels sémaphores du fichier *Cosim-esclave* dans une boucle *if* qui vérifie que la variable *OnTimeMesh* est bien à 1 (ou vrai).

3.3.2 Introduction du modèle de ligne

Une autre modification importante au code original est l'introduction du modèle de ligne à paramètres constants dans l'outil de parallélisation. En effet, l'outil de base de co-simulation ne faisait que transiter les signaux de contrôle de l'instance maître à l'instance esclave, et vice-versa. Or, des fonctions supplémentaires ont dû être intégrées de manière symétrique dans les fichiers source DLL *Cosim-maître* et *Cosim-esclave*.

La première étape fut de permettre l'introduction des données de ligne par l'utilisateur et la lecture de ces dernières dans la procédure d'initialisation des DLLs. Les données qui peuvent être

lues par le modèle DLL doivent être inscrites dans l'attribut *ModelData* du bloc DLL, à la suite du chemin d'accès complet du fichier DLL (tel qu'expliqué à la section 3.2.2). Les données entrées dans ce champ sont passées comme argument dans la requête *req_initialize_new*, suivant exactement le même modèle que les lignes d'EMTP avec les mêmes options. Les variables de ligne à entrer sont les suivantes :

- Nombre de phases (3 ou 6 pour une ligne biterne);
- Modèle sans distorsion (0 ou 1), ligne à conducteurs continuellement transposés (0 ou 1), type de données d'entrée (1, 2 ou 3);
- Mode 0 : Longueur de la ligne, *R' L' C'* (type 1) ou *R' Z_c v* (type 2) ou *R' Z_c τ* (type 3);
- Mode 1 et plus : même format d'entrée;
- Matrice de transformation $n \times n$ (si la ligne n'est pas continuellement transposée, voir section 2.2);
- Vecteurs des tensions triphasées initiales côté *k* et *m* (voir section 3.3.4).

Au point 1, on voit que seules des lignes de 3 ou 6 phases sont acceptées dans l'outil qui a été développé. Il aurait été possible de généraliser le code pour un nombre de phases arbitraire. Or, l'outil a été conçu ainsi pour des raisons d'efficacité et puisque ce sont les lignes les plus utilisées. Au point 3, la variable Z_c correspond à l'impédance caractéristique de la ligne (voir section 2.2). La variable *v* représente la vitesse de l'onde électromagnétique dans la ligne, tandis que τ définit plutôt le temps que prendra l'onde pour parcourir la ligne. Tout comme le modèle de ligne CP programmé dans le logiciel EMTP, l'outil permet plusieurs types d'entrée de données puisqu'ils dépendent tous uniquement des données électriques *R'*, *L'* et *C'*, et peuvent se recalculer aisément. Puis, ces données sont lues et gardées en mémoire dans les objets de ligne de l'instance maître et de l'instance esclave. Elles doivent être entrées de manière séparée et identique dans ces deux instances.

Plusieurs étapes de calcul ont ensuite dû être intégrées aux différentes procédures de l'outil, afin de représenter l'échange de données d'une instance à l'autre. Ces étapes sont résumées dans les points suivants pour chaque procédure. Vous noterez deux changements par rapport aux requêtes précédemment énoncées. Tout d'abord, les procédures ne commencent plus par « req » mais par « dll ». En effet, lorsque le programmeur a indiqué au solveur que le bloc placé sur le schéma est

un bloc DLL avec l'attribut *Part* (voir section 3.2.2), la fonction répondant à cette requête doit commencer par « dll » au lieu de « req ». Deuxièmement, une requête supplémentaire est envoyée aux objets DLL par le solveur, soit *dll_post_initialize_new*. Cette requête est utilisée pour des étapes d'initialisation supplémentaire du code qui pourrait nécessiter les données de simulation du module Fortran *simulation_data*, qui est disponible seulement après la requête *req_initialize_new*.

- *dll_initialize_new* :
    - Lecture des données;
    - Calcul de la résistance du modèle de ligne de la Figure 2-4 et l'inclusion des pertes selon l'équation (2.32).
- *dll_post_initialize_new* :
    - Vérifier que le temps de propagation est plus grand ou égal au pas de temps;
    - Initialisation des tampons de valeurs historiques de courants (i_{kh} pour l'instance maître et i_{mh} pour l'instance esclave) avec courant de ligne nul et tensions des extrémités *k* et *m* égales;
    - Récupération des index de nœud des connecteurs du bloc.
- *dll_put_in_Yn_ss* :
    - Calcul des matrices d'admittances modales propres et mutuelles;
    - Ne sont pas envoyées à la matrice $\mathbf{A}$ du solveur (voir section 3.3.4), mais plutôt utilisées pour calculer les termes i_{kh} et i_{hh} à la fonction *dll_superpose_ss_at_w*.
- *dll_superpose_ss_at_w* :
    - Calcul des termes historiques i_{kh} et i_{mh} à partir des admittances propres et mutuelles, des courants et des tensions de ligne en régime permanent. Ces calculs sont dérivés des équations de la section 2.2 selon le modèle dans le domaine du temps de la Figure 2-4 et de l'inclusion des pertes selon l'équation (2.32).
    - Calcul du tampon des valeurs temporelles des termes historiques i_{kh} et i_{mh} du temps $t = -\tau$ au temps $t = 0$, à chaque pas de temps.

- *dll_put_in_Yn* :
    - Dans le domaine des modes de propagation, la matrice d'admittance temporelle est donnée à l'équation suivante;
    - Multiplication par la matrice de transformation pour passer au domaine des phases et envoi à la matrice **A** avec l'inclusion des pertes selon l'équation (2.32) :

$$\mathbf{Y} = \begin{bmatrix} \frac{1}{Z_c + R'\ell/4} & 0 & 0 \\ 0 & \frac{1}{Z_c + R'\ell/4} & 0 \\ 0 & 0 & \frac{1}{Z_c + R'\ell/4} \end{bmatrix} \tag{3.1}$$

Les fonctions suivantes sont utilisées lors de la simulation dans le domaine du temps, et sont répétées à chaque pas de temps (voir Figure 3-4).

- *dll_put_in_Iaug* :
    - Récupération du courant historique à $t - \tau$, avec interpolation si nécessaire;
    - Conversion vers le domaine des phases et envoi au vecteur **b** du solveur.
- *dll_update_at_t* :
    - Calcul des courants i_k (maître) et i_m (esclave) au pas de temps actuel *t*;
    - Partage des données de courant et de tension entre le maître et l'esclave :

$$\text{Maître à esclave : } Z_c\left(i_{kh}\left(Z_c + R\right) + V_k\right) \tag{3.2}$$

$$\text{Esclave à maître : } Z_c\left(i_{mh}\left(Z_c + R\right) + V_m\right) \tag{3.3}$$

    - Mise à jour des tampons i_{kh} (maître) et i_{mh} (esclave) des courants historiques avec les données partagées.
- *dll_traptoeba_update_at_t*,
  dll_eba_update_at_t,
  dll_ebatotrap_update_at_t :

- Peuvent être utilisées si la mise à jour diffère selon l'intégration;
- Pour l'outil de parallélisation, renvoie à la fonction *dll_update_at_t*.

- *dll_end* :
  - Libération de toutes les variables allouées dynamiquement.

Les actions décrites précédemment, pour chaque fonction des fichiers source *Cosim-maître* et *Cosim-esclave*, viennent se superposer aux appels sémaphores illustrés à la Figure 3-7. D'autres ajouts dans ces fichiers sources seront décrits à la section 3.3.4. Finalement, on note que ces étapes sont quasi-symétriques entre le code « maître » et le code « esclave », à l'exception des différences spécifiquement énoncées ci-haut.

3.3.3 Lignes en parallèle entre deux sous-réseaux

L'outil original de co-simulation permettait de connecter une instance maître à plusieurs esclaves, dans l'éventualité où un réseau contiendrait plusieurs systèmes de contrôle dans des fichiers différents. Or, il n'était pas possible d'avoir plusieurs bus de co-simulation en parallèle entre un maître et un esclave. Ceci permettait de placer plusieurs blocs DLL à l'intérieur d'une même instance esclave faisant référence à un seul bus, afin de disposer les connecteurs de manière plus pratique. De plus, étant donné que l'outil de co-simulation ne partageait que des signaux indépendants de contrôle, il n'y avait aucun avantage à départager ces signaux entre plusieurs bus.

Dans un outil de parallélisation des simulations de puissance, cette particularité devient limitante. En effet, dans les réseaux de tests usuels ainsi que dans les réseaux réels, il est répandu que plusieurs lignes de transmission relient deux ou plusieurs sous-réseaux, comme à l'exemple montré à la Figure 3-8. Des modifications ont donc dû être apportées aux fichiers *Cosim-maître* et *Cosim-esclave*, mais également aux deux fichiers *FMI-maître* et *FMI-esclave*. Ces ajouts ont permis de coupler des réseaux par plusieurs lignes en parallèle, contrairement à certains logiciels.

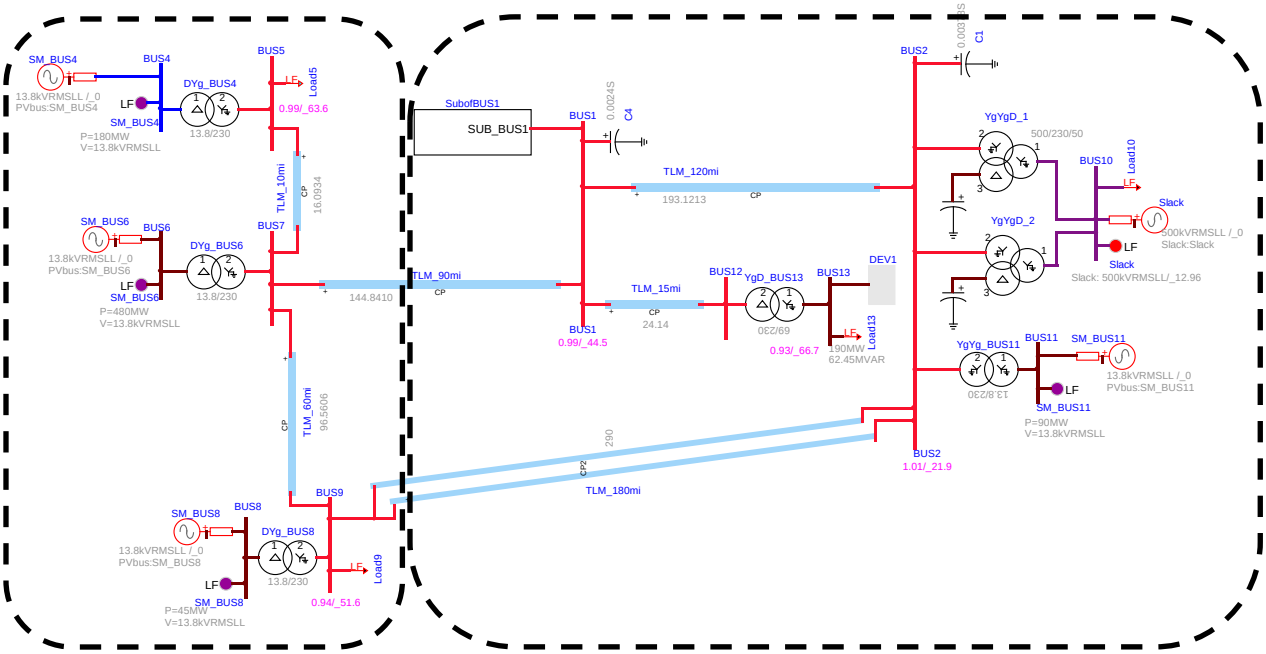

Figure 3-8 Exemple d'un réseau de test avec plusieurs lignes de transmission en parallèle

Dans l'outil original de co-simulation, la manière dont les variables étaient gérées dans les fichiers *Cosim-esclave* et *FMI-esclave* était différente de leurs homologues de l'instance maître. En effet, tous les blocs DLL faisaient partie du même objet global *Global_Slave_Instance*, donc partagés entre toutes les fonctions. Ainsi, lorsque le logiciel EMTP détectait un autre bloc DLL dans l'instance esclave, un nouvel objet n'était pas créé et surtout, un nouvel espace mémoire n'était pas alloué pour le bus de co-simulation. Donc, à l'étape de création du bus dans la fonction *dll_initialize_new* du DLL *Cosim-esclave*, une vérification était faite sur l'objet global pour savoir si le code traite le premier bloc DLL. Si c'est le cas, l'espace mémoire était alloué, sinon rien n'était fait.

Ainsi, la première modification pour permettre plusieurs lignes en parallèle fut de remplacer cet objet global par un objet local, instancié indépendamment pour chaque bus de co-simulation. L'allocation de mémoire est également faite pour tous les blocs DLL de l'instance esclave en délaissant la vérification à l'initialisation.

Puis, des fonctions ont également dû être modifiées dans les fichiers DLL *FMI-maître* et *FMI-esclave*. Dans le fichier *FMI-esclave*, une structure a été mise en place pour traiter un tableau de bus de co-simulation au lieu de n'accepter qu'un seul bus par instance. Ce passage de variable

simple à tableau est semblable à ce qui avait été fait au niveau du maître, où plusieurs bus de co-simulation étaient acceptés. Ces éléments ont été ajoutés à l'intérieur de la fonction *CreateFMIInstance* (voir section 3.2.3), dans la procédure *addFMI*, chargée de l'allocation de mémoire selon la grandeur des bus de co-simulation. Finalement, une autre modification a dû être réalisée dans le fichier DLL *FMI-maître*, à la fonction *FMIInstantiateSlave* (voir section 3.2.3). Pour chaque esclave, le maître y fait une remise à zéro des sémaphores pour initialiser ces variables.

Suivant ces modifications et ajouts, la parallélisation avec plusieurs lignes en parallèle devient possible. Il faut toutefois que l'usager désigne différemment les bus de co-simulation pour chaque lien afin que les structures expliquées ci-haut fonctionnent. Ces données sont entrées dans l'attribut *ModelData*. Le schéma de communication d'un exemple avec différents fichiers de code avec des bus de co-simulation en parallèle vers la même instance est illustré à la Figure 3-9.

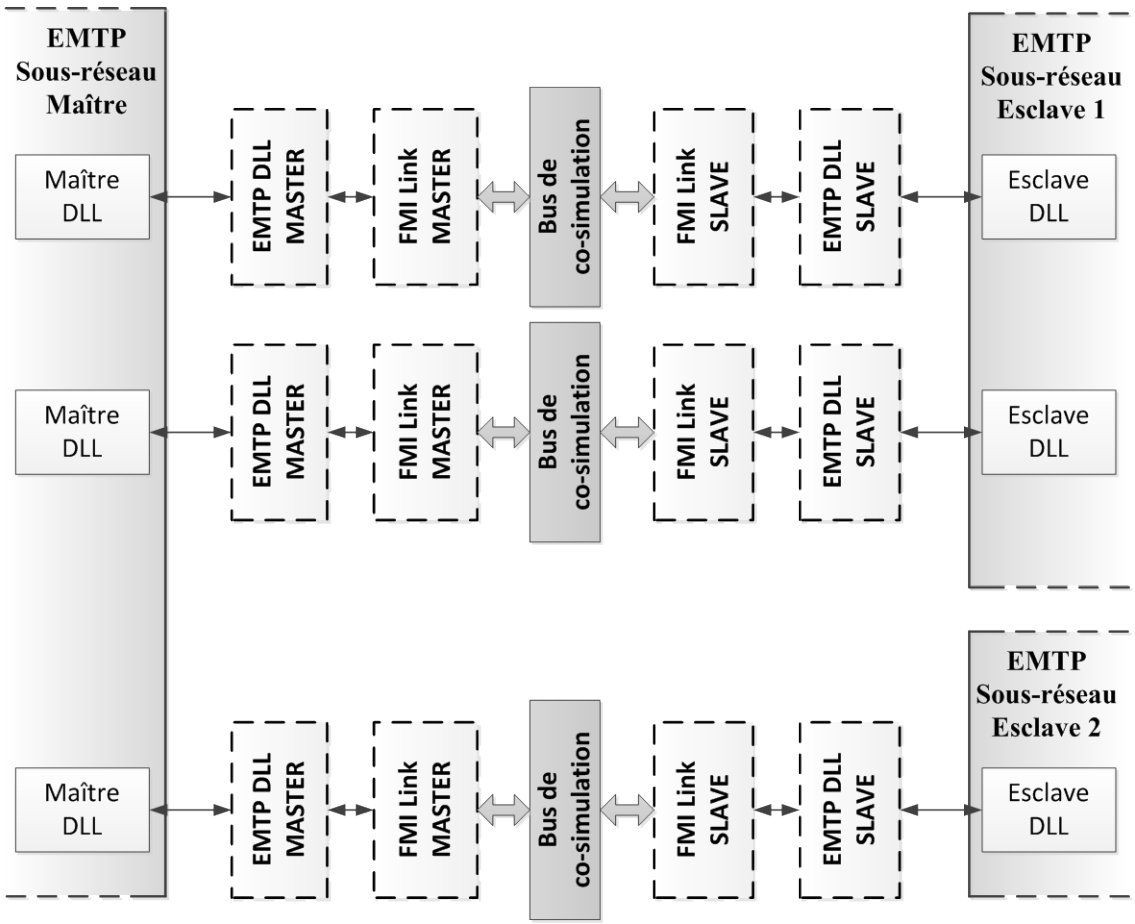

Figure 3-9 Communication de trois sous-réseaux par des bus de co-simulation en parallèle

3.3.4 Initialisation des sous-réseaux

Tel que mentionné à la section 2.3.3, d'autres logiciels de simulation des réseaux électriques sont parvenus, dans les années précédentes, à effectuer une simulation parallèle sur une architecture multi-cœurs. Or, ces simulations présentent plusieurs limitations, dont l'impossibilité d'initialiser les différents sous-réseaux à partir d'un calcul en régime permanent. Ainsi, le logiciel doit attendre que le réseau se stabilise à son régime permanent au début de chaque simulation, ce qui est pénalisant si le but des travaux est l'accélération des simulations.

Le logiciel EMTP initialise automatiquement ses simulations dans le domaine du temps par un calcul effectué en régime permanent. Or, cette initialisation ne peut se faire lorsque le réseau est découplé en sous-réseaux par l'outil de parallélisation présenté ici. L'initialisation doit donc se faire en plusieurs étapes. Tout d'abord, l'utilisateur de l'outil doit faire un calcul en régime permanent sur le réseau complet afin d'obtenir les phaseurs de tension à chaque bout de la ligne de transmission servant au découplage. Cette première étape est illustrée à la Figure 3-10.

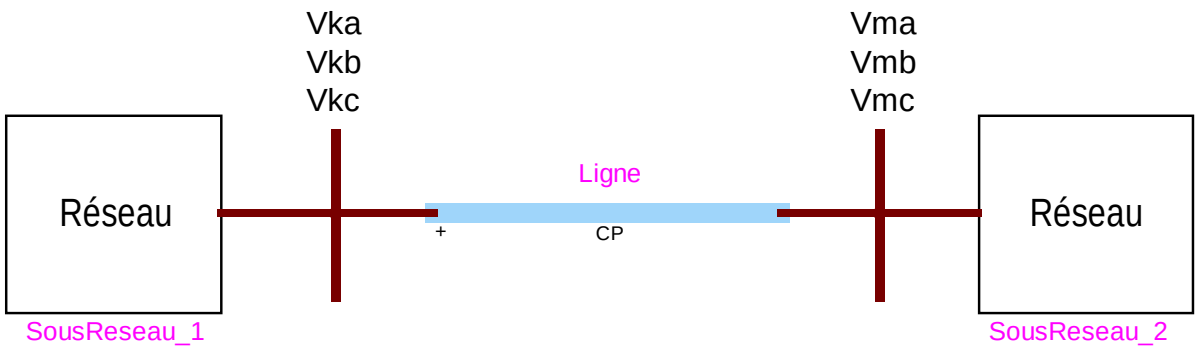

Figure 3-10 Récupération des phaseurs de tension au régime permanent du réseau complet

Ensuite, la deuxième étape consiste à récupérer ces tensions pour les introduire suivant les données de ligne dans les blocs DLL avec, tel que mentionné au début de la section 3.3.2. Ainsi, la lecture des données par le fichier *Cosim-maître* ou *Cosim-esclave* est réalisée en entrant manuellement les phaseurs dans le *ModelData* des blocs maître et esclave. Toutefois, la lecture pourrait aussi être réalisée automatiquement, car le logiciel permet la lecture des tensions en régime permanent via ses fichiers de sortie. Cet élément fait partie des possibles améliorations futures discutées dans la conclusion.

Une fois les données des tensions en régime permanent parvenues aux fichiers *Cosim-maître* et *Cosim-esclave*, l'outil de parallélisation participe à l'initialisation par le régime permanent à la manière que celle de tous les équipements des sous-réseaux. C'est pourquoi, dans l'outil de parallélisation, le modèle en régime permanent de la ligne de transmission n'est pas un modèle PI-exact mais plutôt une source de tension triphasée indépendante aux extrémités du maître et de l'esclave (Figure 3-11). Ainsi, le calcul en régime permanent automatique de chaque sous-réseau sera identique à celui du réseau complet et la simulation sera stable dès le premier pas de temps, tel que confirmé par les tests de validation qui sont montrés à la section 4.1.

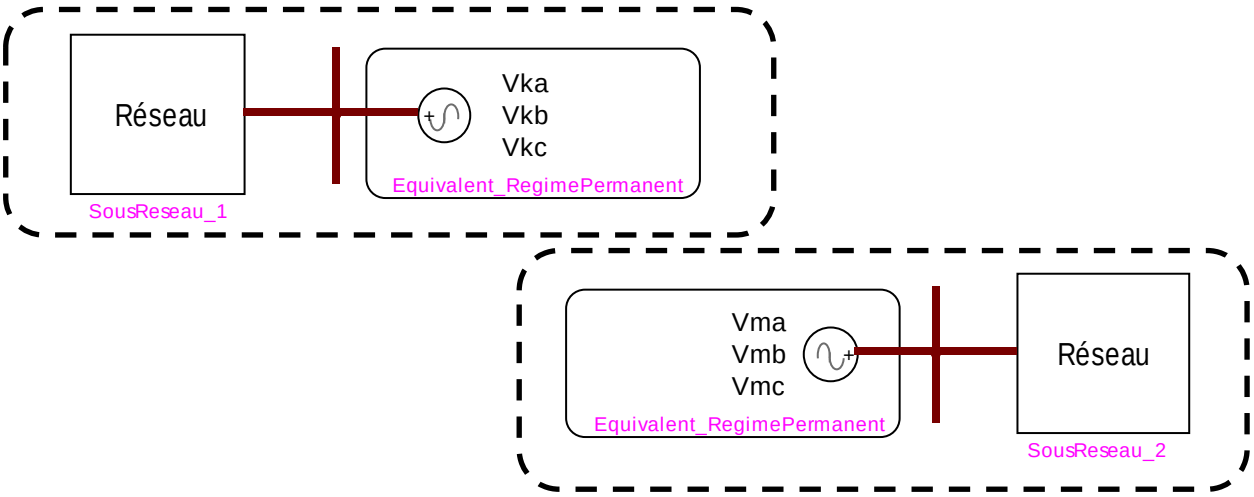

Figure 3-11 Instance maître et instance esclave en régime permanent

L'introduction d'une source de tension en régime permanent est effectuée de manière symétrique à l'intérieur des fonctions suivantes :

- *dll_put_voltage_row_equations* :
  - Indication au solveur de réserver l'espace dans la matrice **A** ;
- *dll_put_in_Yaug_ss* :
  - Entrée des données dans la matrice **A** pour former le système d'équation (3.4);
- *dll_put_in_Iaug_ss* :
  - Entrée des phaseurs de tension dans le vecteur **b** sous forme de nombre complexe. Le système d'équations résultant est le suivant :

$$\begin{bmatrix} & & & \dots & 1 & & \\ & \mathbf{Y_n} & & \dots & & 1 & \\ & & & \dots & & & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & & & \dots & 0 & & \\ & 1 & & \dots & & 0 & \\ & & 1 & \dots & & & 0 \end{bmatrix} \times \overline{\mathbf{V}}_{\mathbf{noeuds}} = \begin{bmatrix} \\ \overline{\mathbf{I}}_{\mathbf{n}} \\ \\ \dots \\ \overline{\mathbf{V}}_{\mathbf{a\_RP}} \\ \overline{\mathbf{V}}_{\mathbf{b\_RP}} \\ \overline{\mathbf{V}}_{\mathbf{c\_RP}} \end{bmatrix} \tag{3.4}$$

3.3.5 Optimisation de la vitesse d'exécution de l'outil

Un travail supplémentaire a été fait à plusieurs niveaux dans le but d'améliorer la vitesse d'exécution de l'outil de parallélisation. Ces améliorations sont faites uniquement sur les opérations qui sont répétées à chaque pas de temps, étant donné que le domaine du temps consomme la vaste majorité du temps de simulation.

Tout d'abord, comme pour tous les modèles du logiciel EMTP, toutes les matrices (admittance, transformation, etc.) sont stockées sous forme de vecteurs. Étant donné que les opérations mathématiques sont exécutées à l'intérieur d'une seule boucle *for*, ces dernières sont effectuées plus rapidement lors de l'exécution. De plus, étant donné les nombreuses conversions entre domaine modal et le domaine des phases, les opérations de multiplication de matrices avec un vecteur et d'inversion de matrice s'effectuent plusieurs fois par pas de temps. Il est donc avantageux d'optimiser ces opérations qui demandent beaucoup de temps. Ces opérations ont donc été préprogrammées pour le type de lignes disponible dans l'outil, soit standard (3 phases) ou biterne (6 phases). Elles utilisent la méthode des cofacteurs pour l'inversion de la matrice. Cette méthode réduit la flexibilité quant au nombre possible de phases des lignes. Or, les cas multiphasés sont extrêmement rares; on favorise donc le gain en performance au détriment de cette flexibilité.

Une autre opération coûteuse en temps et qui est répétée à chaque pas de temps est la rotation du tampon d'historique de courant. En effet, à chaque pas de temps, le courant actuel i_k (côté maître) ou i_h (côté esclave) sortant à chaque bout de la ligne est gardé en mémoire dans un tampon pour être réutilisé plus tard, au temps $t + \tau$. De plus, le courant qui a été enregistré à un temps $t - \tau$ doit être utilisé pour calculer le courant actuel de l'instance maître et de l'instance esclave. Toutes les données contenues dans le tampon doivent donc être décalées pour effacer l'entrée la plus

ancienne et libérer l'espace à la fin pour le courant actuel. La grandeur de ce tampon dépend de la longueur de la ligne et du pas de temps choisi :

$$N_{termes\ historiques} = entier\left(\frac{\tau}{\Delta t}\right) \tag{3.5}$$

Ce nombre de termes peut grandement varier de quelques dizaines à plusieurs milliers, c'est pourquoi la rotation du tampon doit être optimisée. Ainsi, la fonction C *memmove* est utilisée pour cette opération, car aucune boucle n'est nécessaire et c'est une fonction de très bas niveau informatique. Elle sert spécifiquement à déplacer un nombre d'octets d'une adresse mémoire à une autre en une seule opération.

Finalement, le compilateur a une grande part à jouer dans la vitesse d'exécution d'un logiciel ou d'une librairie de fonctions. Le compilateur utilisé dans le présent projet, *Microsoft Visual Studio 2010*, comporte une panoplie d'options d'optimisation selon le type de code et les besoins. Les options évaluées ici sont les suivantes :

- /O1 : Créé le code le plus petit dans la majorité des cas;
- /O2 : Créé le code le plus rapide dans la majorité des cas;
- /Ox : Optimisation complète, favorise la vitesse d'exécution au détriment de la grosseur du code;
- /GL : Active l'optimisation sur le programme de manière globale. Si cette option n'est pas activée, l'optimisation est faite sur les modules individuels;
- /LTCG : Pour une meilleure performance de compilation, doit être appelé avec /GL. Indique l'optimisation du code à l'étape de la liaison (*linking*).

Plusieurs combinaisons d'optimisations ont été testées selon les cinq cas suivants :

- Cas 1 : Compilation en mode *Debug*;
- Cas 2 : Compilation en mode *Release*, /O2, /GL, /LTCG;
- Cas 3 : Compilation en mode *Release*, /O2;
- Cas 4 : Compilation en mode *Release*, /O1;
- Cas 5 : Compilation en mode *Release*, /Ox.

Ces cinq combinaisons ont été testées sur le cas montré à la Figure 3-8, découplé en deux sous-réseaux par les lignes *TLM_90mi* et *TLM_60mi*. Les résultats de performance, présentés au Tableau 3-2, permettent de déterminer l'optimisation à utiliser pour les tests de performance présentés au Chapitre 4.

Tableau 3-2 Détermination des options d'optimisation du compilateur

Temps (s)	**Cas 1**	**Cas 2**	**Cas 3**	**Cas 4**	**Cas 5**
Optimisations	*Debug*	/O2, /GL, /LTCG	/O2	/O1	/Ox
Maître	0,93601	0,65520	0,59280	0,56160	0,57720
Esclave	0,96721	0,63960	0,62400	0,68640	0,63960

L'évaluation de la performance est discutée plus en détails au Chapitre 4, or nous évaluons ici plutôt la qualité des optimisations proposées en évaluant seulement l'instance (maître ou esclaves) la plus lente, qui correspond au temps réel de la simulation. Pour chaque cas, les quatre librairies DLL ont été compilées avec l'optimisation mentionnée au Tableau 3-2.

Tout d'abord, on constate que la simple utilisation du mode de compilation *Release* augmente grandement les performances, même sans optimisation supplémentaire. Puis, en comparant les cas 2 et 3, on remarque que l'optimisation globale (/GL) couplée à l'activation de /LTCG n'améliore pas les performances et crée même des avertissements à la liaison. Finalement, les cas 3, 4 et 5 permettaient de comparer les optimisations générales /O1, /O2 et /Ox. Normalement, /O2 compile un programme rapide, mais il est possible que la réduction de la taille du code par /O1 améliore davantage la performance, par exemple dans une situation de mouvements abusifs de pages de mémoire (*paging*). L'optimisation /Ox, plus récent ajout au compilateur utilisé, est une optimisation globale qui prend en compte cet aspect. Selon les résultats, on constate que pour ce projet, il n'y a pas une grande différence entre ces options, même si c'est l'optimisation /O2 qui est légèrement meilleure. Afin d'obtenir les meilleurs résultats sur la plupart des processeurs, nous conservons donc cette optimisation pour l'ensemble des tests de performance présentés au Chapitre 4.

CHAPITRE 4 PRÉSENTATION ET ANALYSE DES RÉSULTATS

Les tests de validations de l'outil sont réalisés sur deux réseaux tests. Puis, la performance de l'outil est étudiée à l'aide de trois réseaux et leurs variantes. Les résultats sont ensuite analysés selon leur configuration et les chronomètres détaillés des simulations. Des considérations telles que le temps de communication entre les sous-réseaux, l'impact des modèles mathématiques utilisés et la configuration du réseau de base sont également discutés. L'ordinateur utilisé possède un processeur à quatre cœurs *Intel* i7-2640M à 2.80 GHz et 6 GB de mémoire vive.

4.1 Validation de l'outil proposé

Deux réseaux ont été choisis pour valider le bon fonctionnement de l'outil de parallélisation. Les deux permettent d'étudier des phénomènes et de tester des fonctionnalités différentes. Les essais de validité à l'aide de ces réseaux ont contribué à réaliser les fonctionnalités actuelles de l'outil, comme présenté dans les sections suivantes.

4.1.1 Réseau « 230 kV »

Le premier cas de validation proposé est un réseau qui présente deux possibilités naturelles de découpage en sous-réseaux qui sont montrées à la Figure 4-1. Chacun de ces deux cas présente une particularité intéressante qui a nécessité l'ajout des caractéristiques décrites à la section 3.3. Dans les deux configurations de découpage testées, on remarque la présence de deux lignes de transmission entre les deux sous-réseaux. Ce sont ces cas qui ont validé les modifications décrites à la section 3.3.3 à propos des bus de co-simulation multiples entre une instance maître et une instance esclave. De plus, lors du test de validation de la configuration #1, un défaut fugitif au début de la ligne *TLM_120mi*, suivi de l'ouverture de cette dernière, a été introduit dans la simulation afin de tester l'erreur transitoire de l'outil. Ainsi, au moment du défaut, le solveur a dû effectuer un changement temporaire du mode d'intégration vers la méthode d'Euler implicite. Cet essai a confirmé que l'outil est fonctionnel dans les deux modes d'intégration (trapézoïdal et Euler) et lors du passage de l'un à l'autre durant une simulation, tel qu'expliqué à la section 3.3.1. Finalement, la configuration de découpage #2 comporte un point de coupure consistant en une ligne biterne (*TLM_180mi*). Le modèle de ligne biterne a ainsi été validé, comme mentionné à la section 3.3.2.

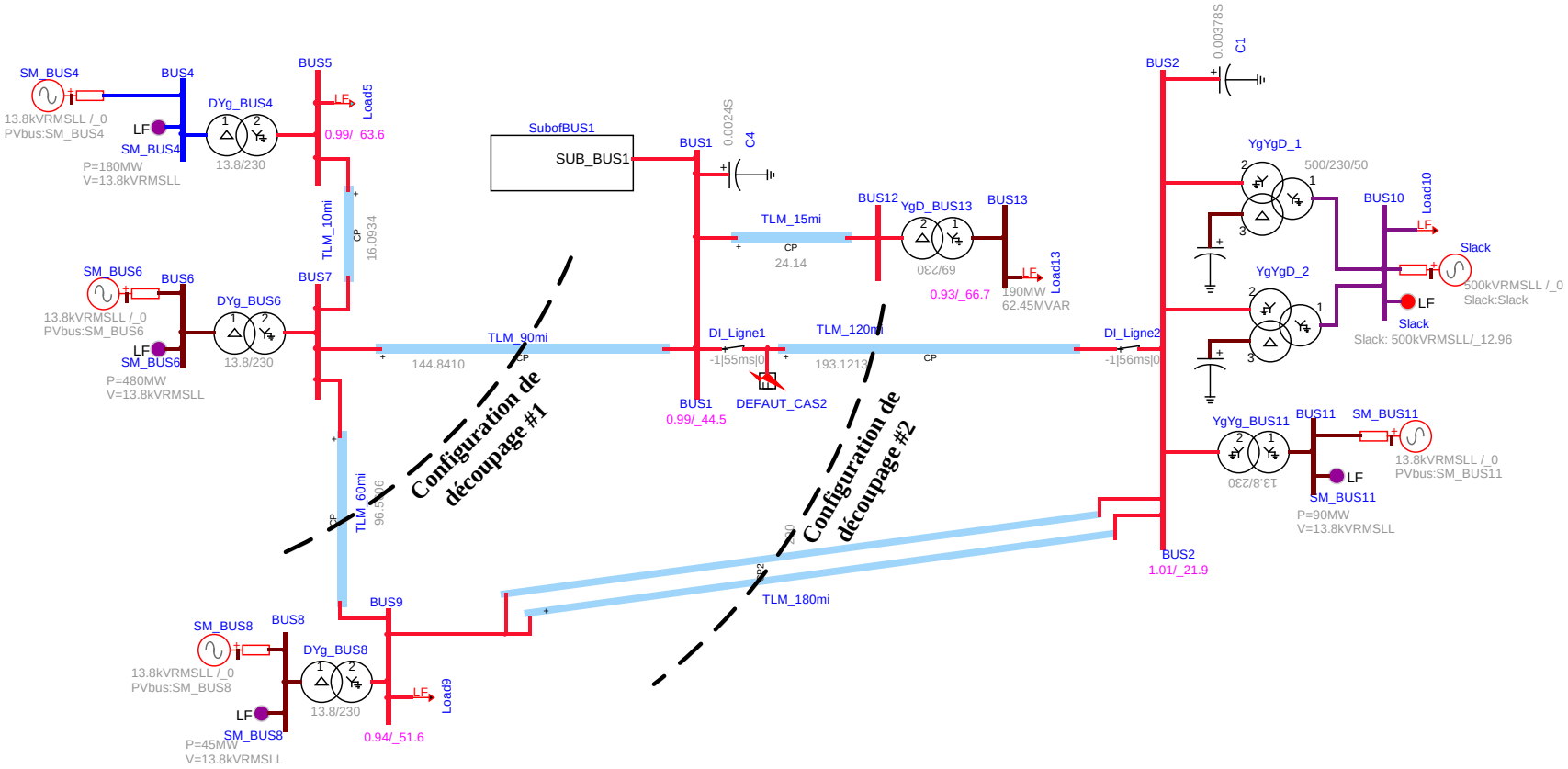

Figure 4-1 Réseau de validation « 230 kV »

4.1.1.1 Configuration de découpage #1

Les résultats de validation montrés ici ont été réalisés avec un défaut de ligne à 50 ms suivi de l'ouverture de la ligne *TLM_120mi* à 55 ms (extrémité *BUS 1*) et 60 ms (extrémité *BUS 2*). Le défaut est éliminé et la ligne est remise sous tension à 150 ms.

Les résultats obtenus avec l'outil sont superposés à ceux obtenus lors la simulation est réalisée par une seule instance (cas de référence). De plus, pour chaque comparaison, un graphique représentant la différence absolue entre ces deux courbes est montré. Finalement, les tensions de phase A à chaque point de coupure sont utilisées comme valeurs électriques de comparaison, soit aux bus #1 (Figure 4-2), #7 (Figure 4-3) et #9 (Figure 4-4).

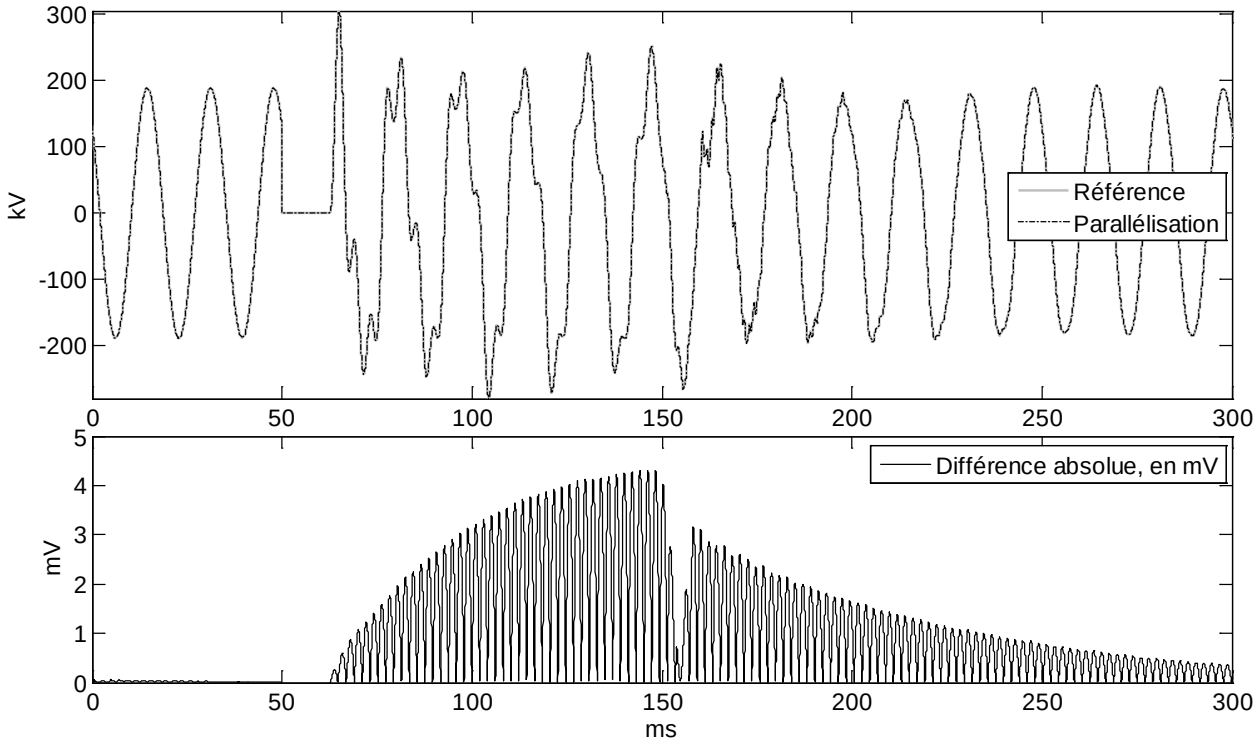

Figure 4-2 Réseau de validation « 230 kV », cas de découpage #1, tension BUS1

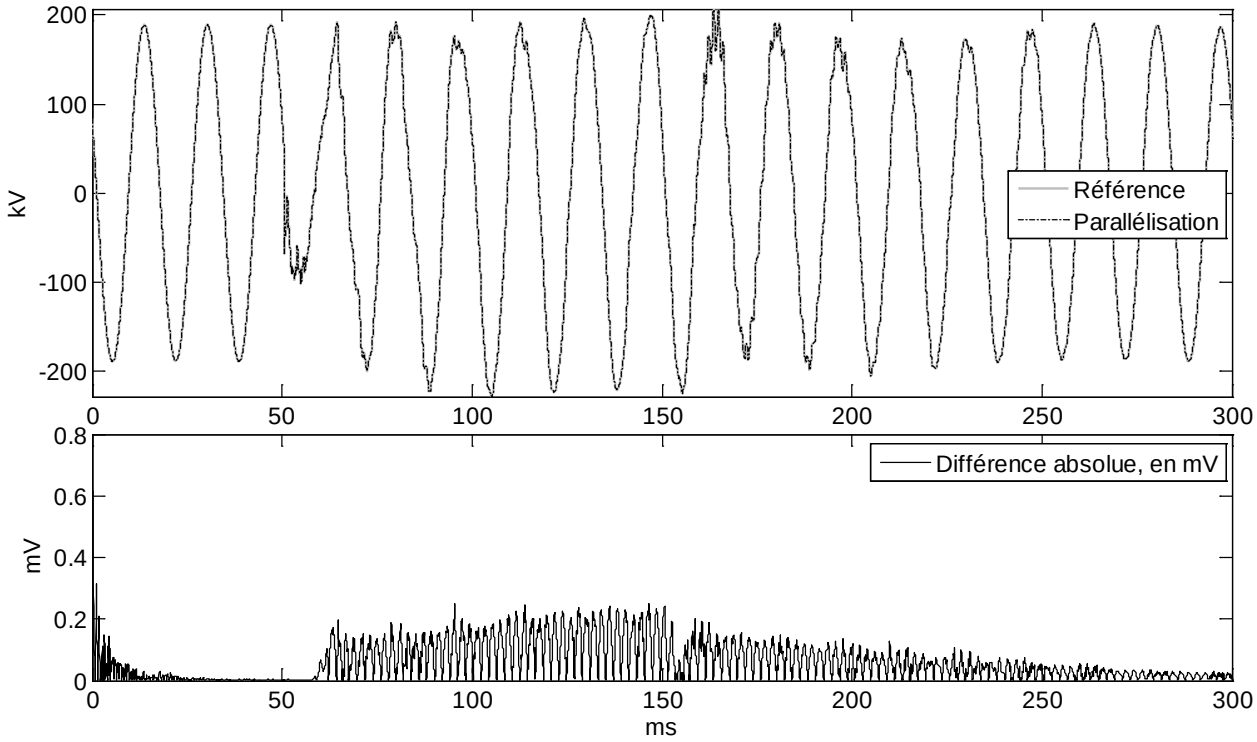

Figure 4-3 Réseau de validation « 230 kV », cas de découpage #1, tension BUS7

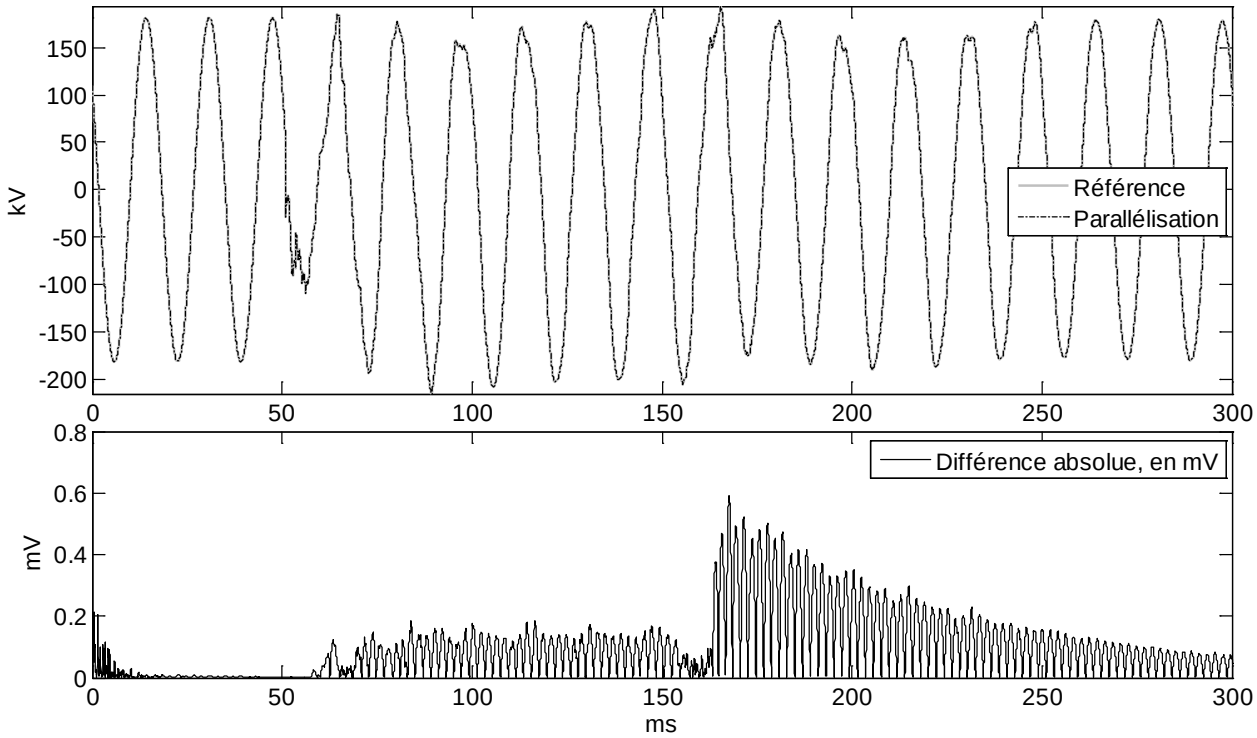

Figure 4-4 Réseau de validation « 230 kV », cas de découpage #1, tension BUS9

On voit tout d'abord que l'erreur introduite par l'utilisation de l'outil est négligeable. En effet, celle-ci oscille entre 10^{-6} et 10^{-3} V sur des signaux de l'ordre de 10^{5} V, soit une erreur relative atteignant au maximum 5×10^{-7} %. De manière générale dans l'ensemble des tests de validation, l'erreur augmente lors d'événements transitoires pour diminuer lors du retour à un régime permanent. Aussi, plus la mesure est proche du point de défaut (ex. Figure 4-2), plus l'erreur est importante. Les sources possibles de ces erreurs transitoires peuvent être reliées aux effets variables du solveur sur un réseau complet et par rapport au même réseau découplé, par exemple avec une utilisation différente de la méthode d'intégration Euler implicite (Backward Euler) par le solveur.

4.1.1.2 Configuration de découpage #2

Les valeurs électriques de validation de l'outil sont les tensions de phase A aux points de coupure des sous-réseaux, soit aux bus #1 (Figure 4-5), #2 (Figure 4-6) et #9 (Figure 4-7).

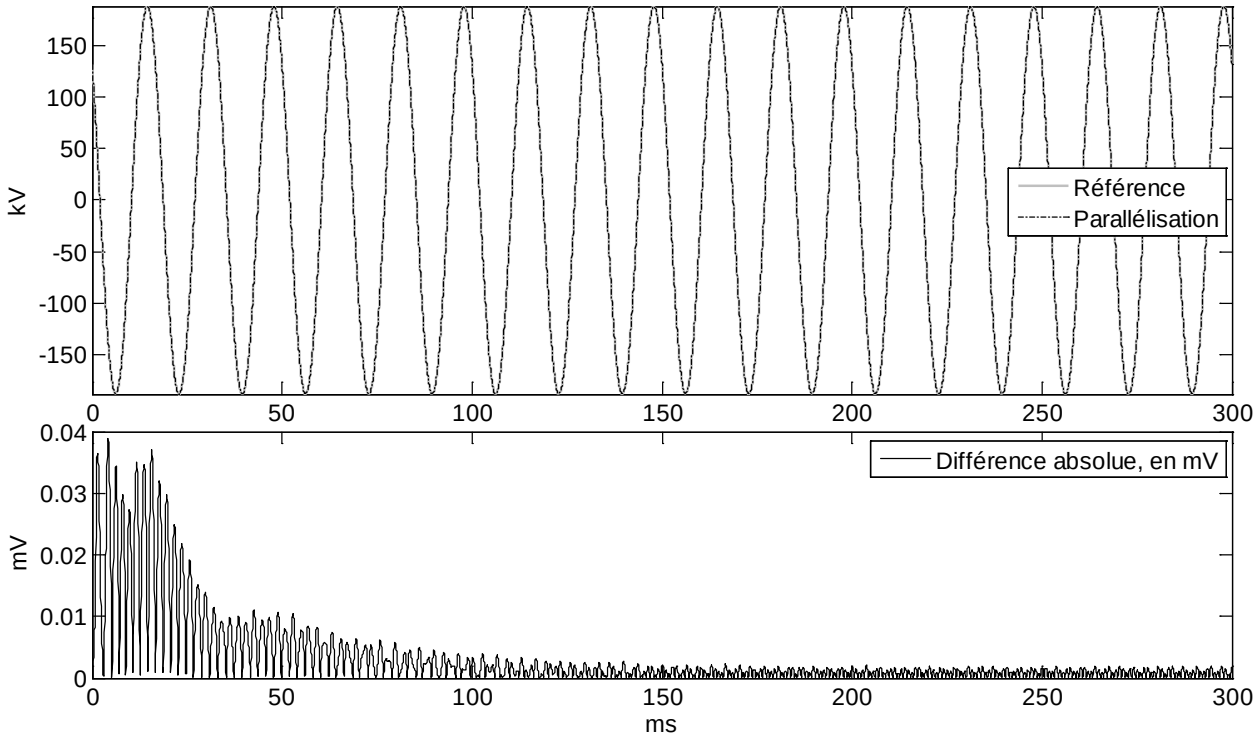

Figure 4-5 Réseau de validation « 230 kV », cas de découpage #2, tension BUS1

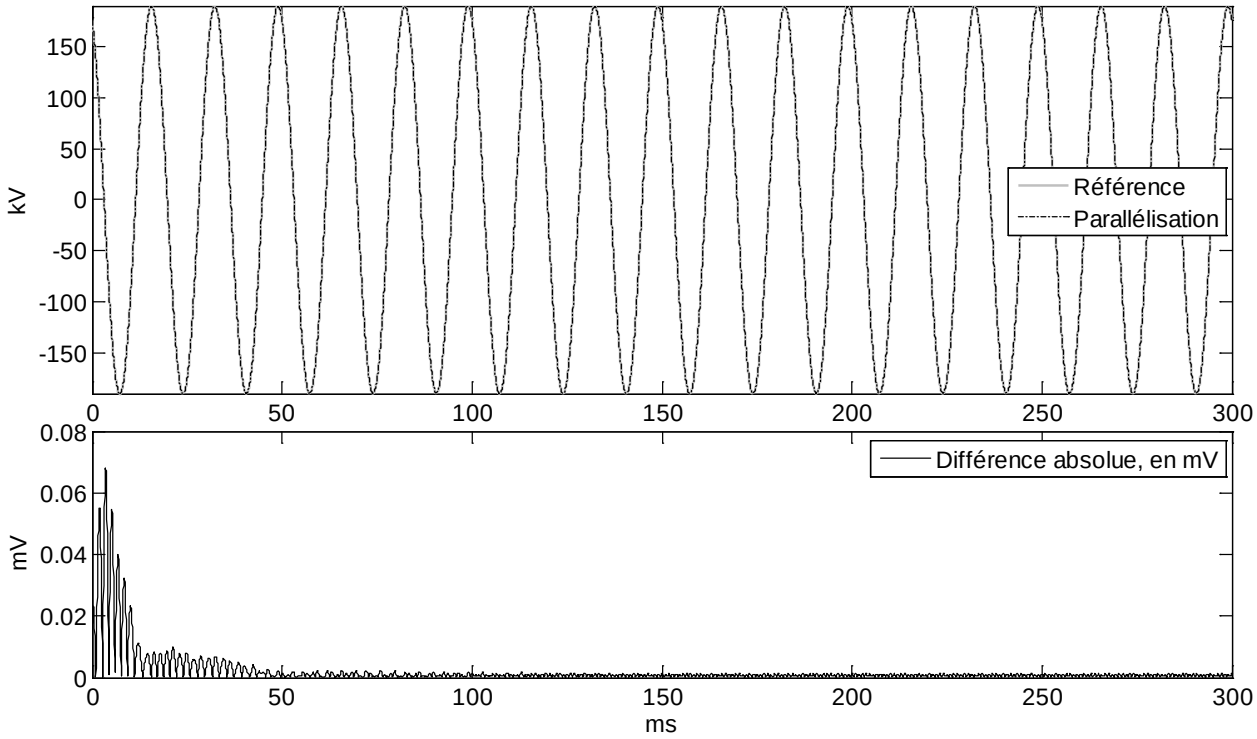

Figure 4-6 Réseau de validation « 230 kV », cas de découpage #2, tension BUS2

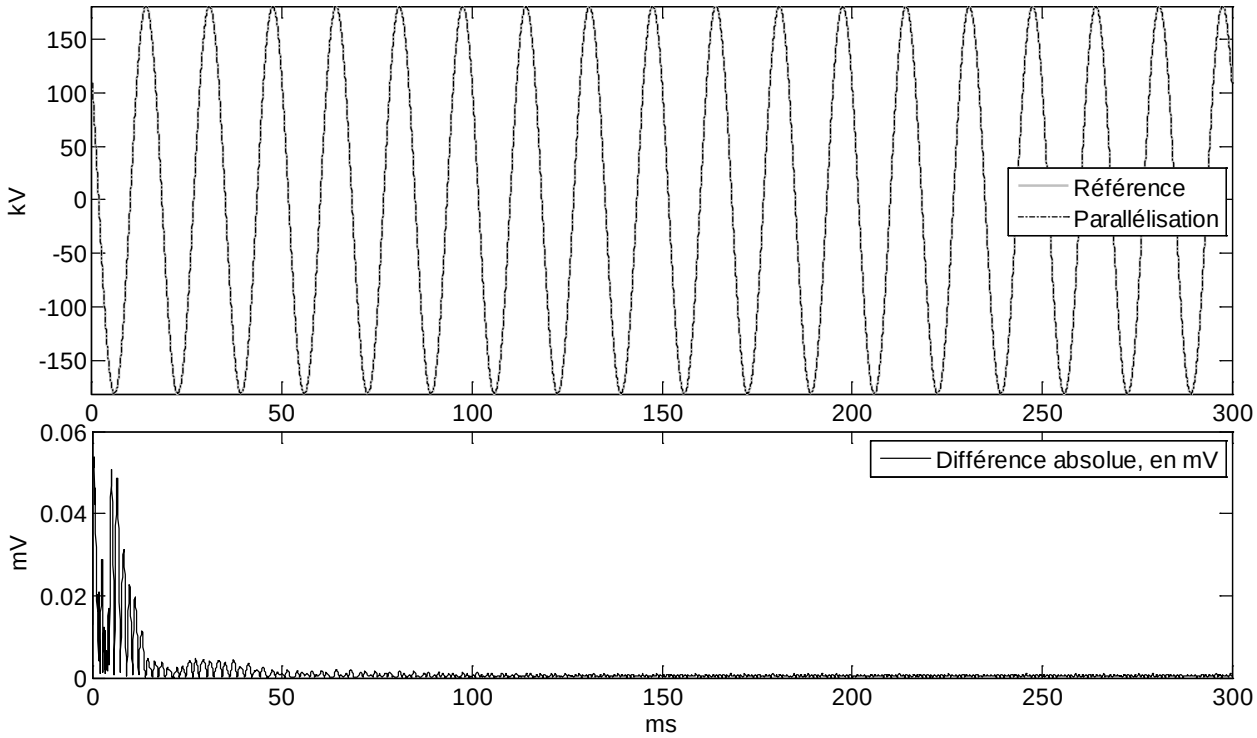

Figure 4-7 Réseau de validation « 230 kV », cas de découpage #2, tension BUS9

On obtient encore une fois une erreur négligeable avec l'utilisation de l'outil. Après l'initialisation, une période de quelques millisecondes présente une erreur de l'ordre de 10^{-5} V, mais après stabilisation l'erreur est réduite encore davantage aux alentours de 10^{-6} V, toujours pour un système avec une tension phase-terre de 187 kV.

Les deux cas de découpage présentés pour le réseau « 230 kV » ont permis de valider l'utilisation du modèle de ligne standard et du modèle de ligne biterne. L'utilisation de plusieurs bus de co-simulation en parallèle entre une même paire d'instances maître-esclave est également vérifiée et validée.

4.1.2 Réseau « Kundur » à trois machines synchrones

L'initialisation des machines synchrones, connectées presque directement au bus de co-simulation de leur sous-réseau, est l'objet de ce test de validation. En effet, si les machines ne sont pas correctement initialisées au temps t=0 de la simulation, des oscillations de puissance peuvent en résulter et subsister pendant une longue durée, ce qui constitue une perte de temps de calcul pour les simulations à effectuer. Le second cas de validation, tiré de [40], comporte trois machines synchrones avec leurs systèmes de régulation et une charge centrale. Le réseau ainsi que le découpage en sous-réseaux est illustré à la Figure 4-8. Les résultats de validation comportent les courants (Figure 4-9, Figure 4-11 et Figure 4-13) et les tensions (Figure 4-10, Figure 4-12 et Figure 4-14) à la sortie de chaque machine synchrone.

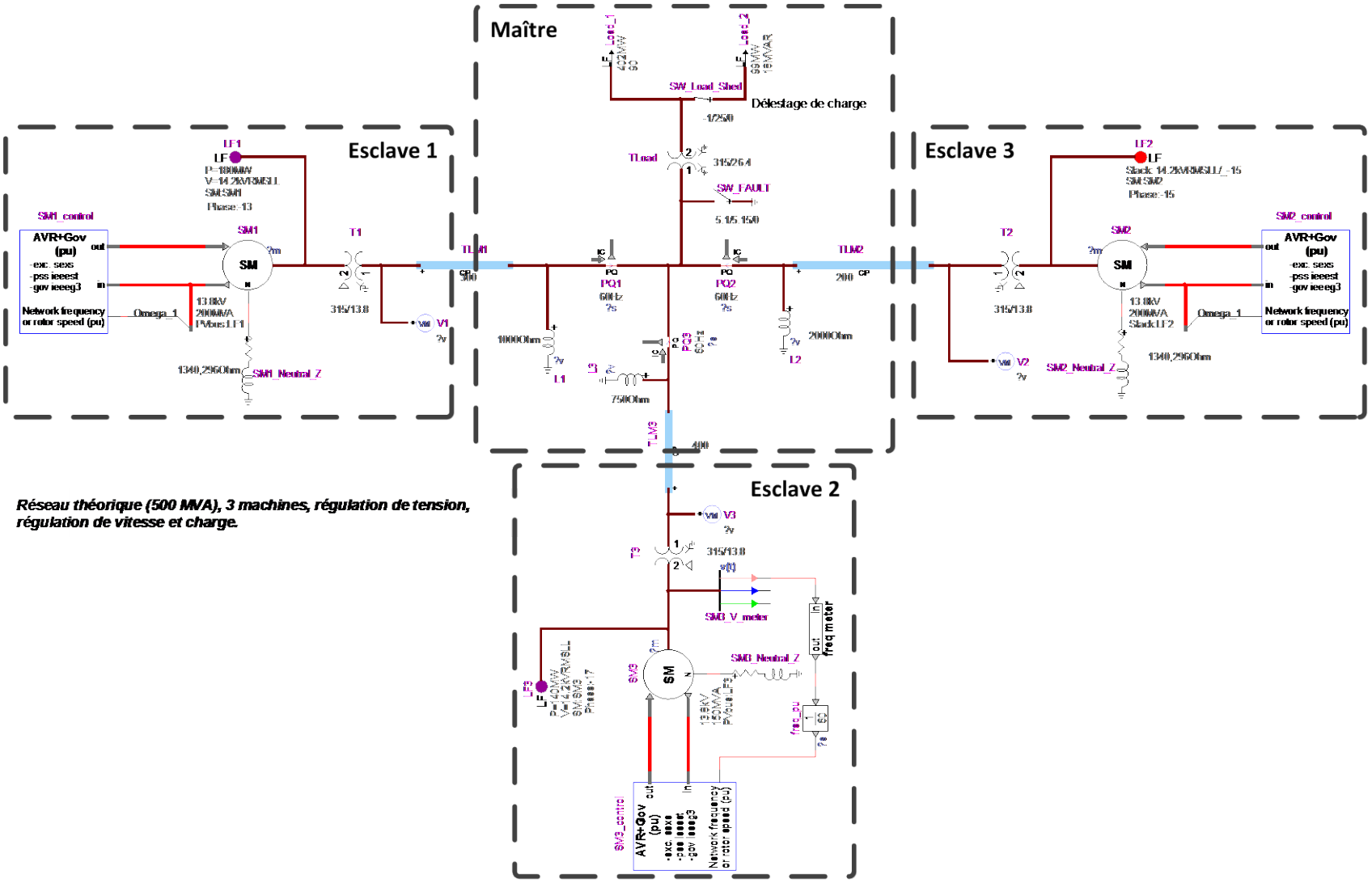

Figure 4-8 Réseau de validation « Kundur »

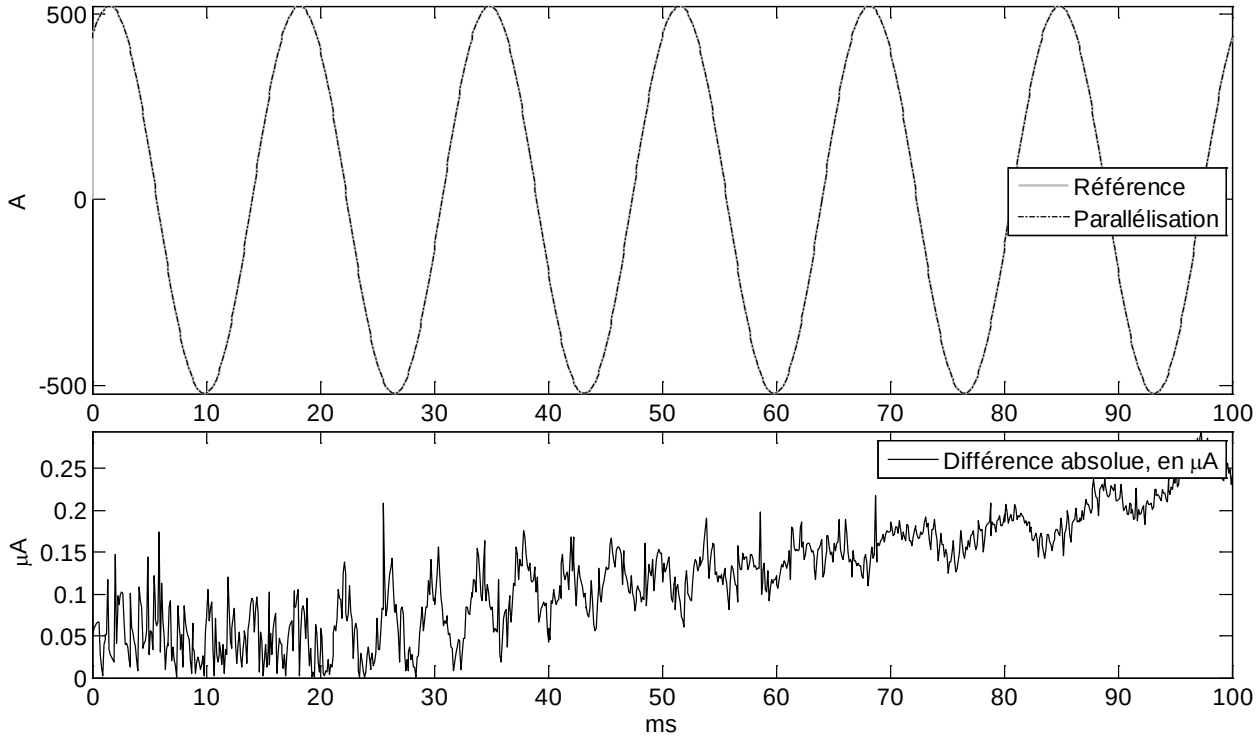

Figure 4-9 Réseau de validation « Kundur », courant machine *SM1*

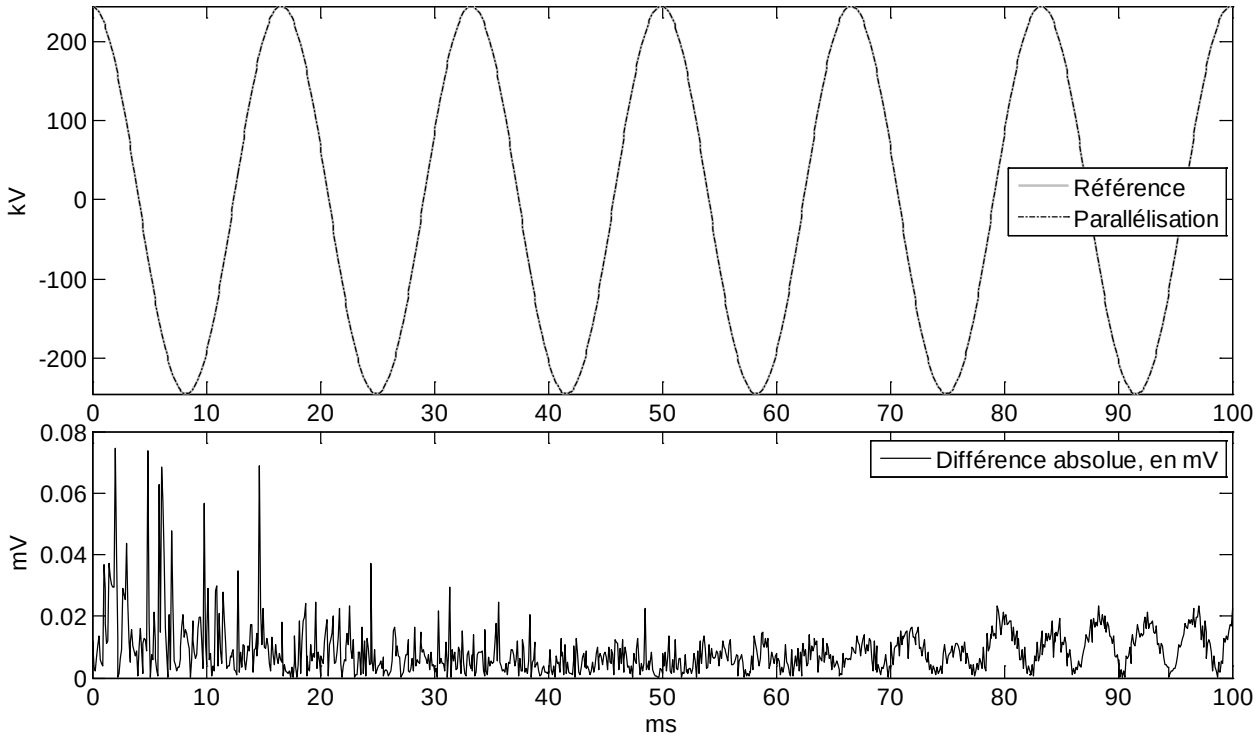

Figure 4-10 Réseau de validation « Kundur », tension machine *SM1*

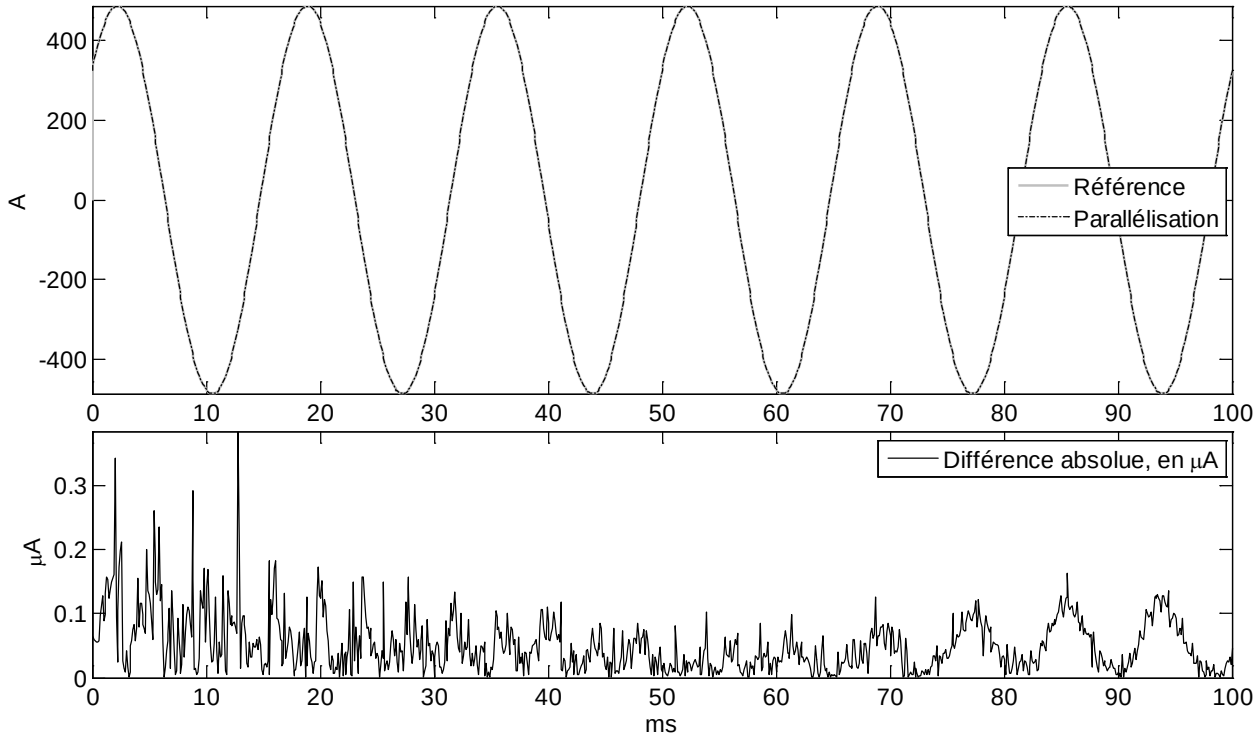

Figure 4-11 Réseau de validation « Kundur », courant machine *SM2*

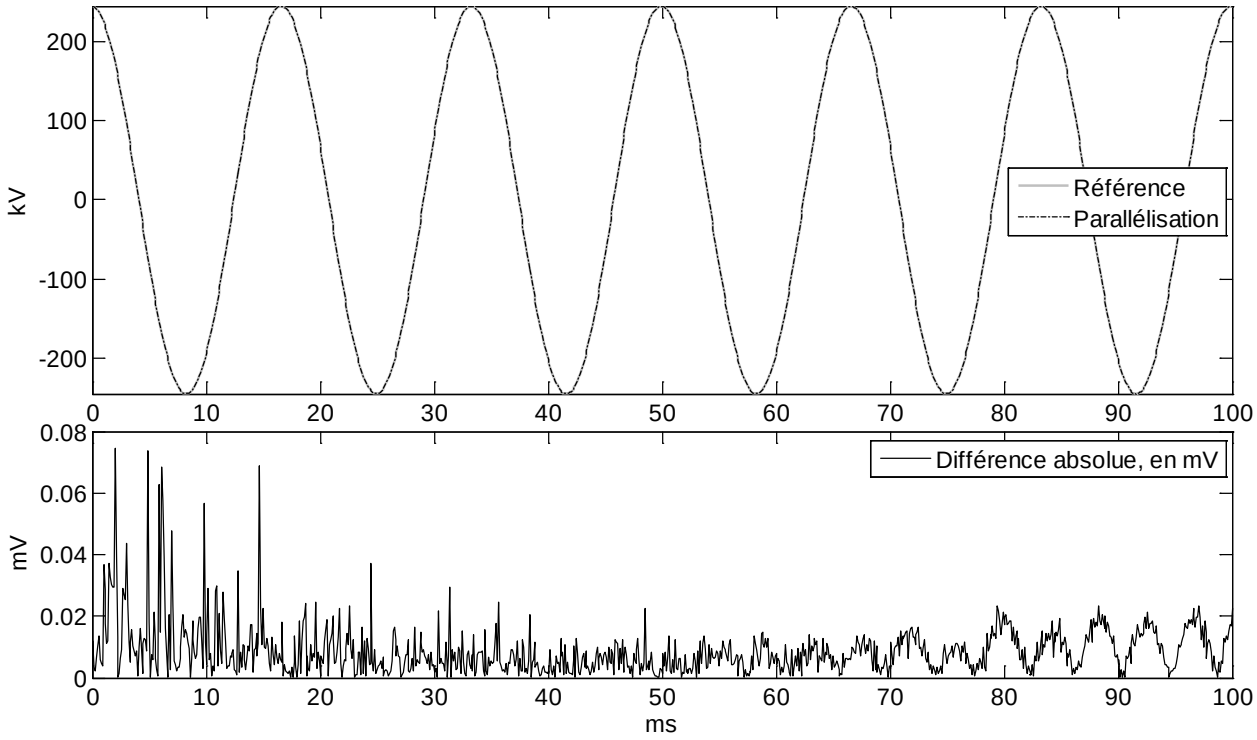

Figure 4-12 Réseau de validation « Kundur », tension machine *SM2*

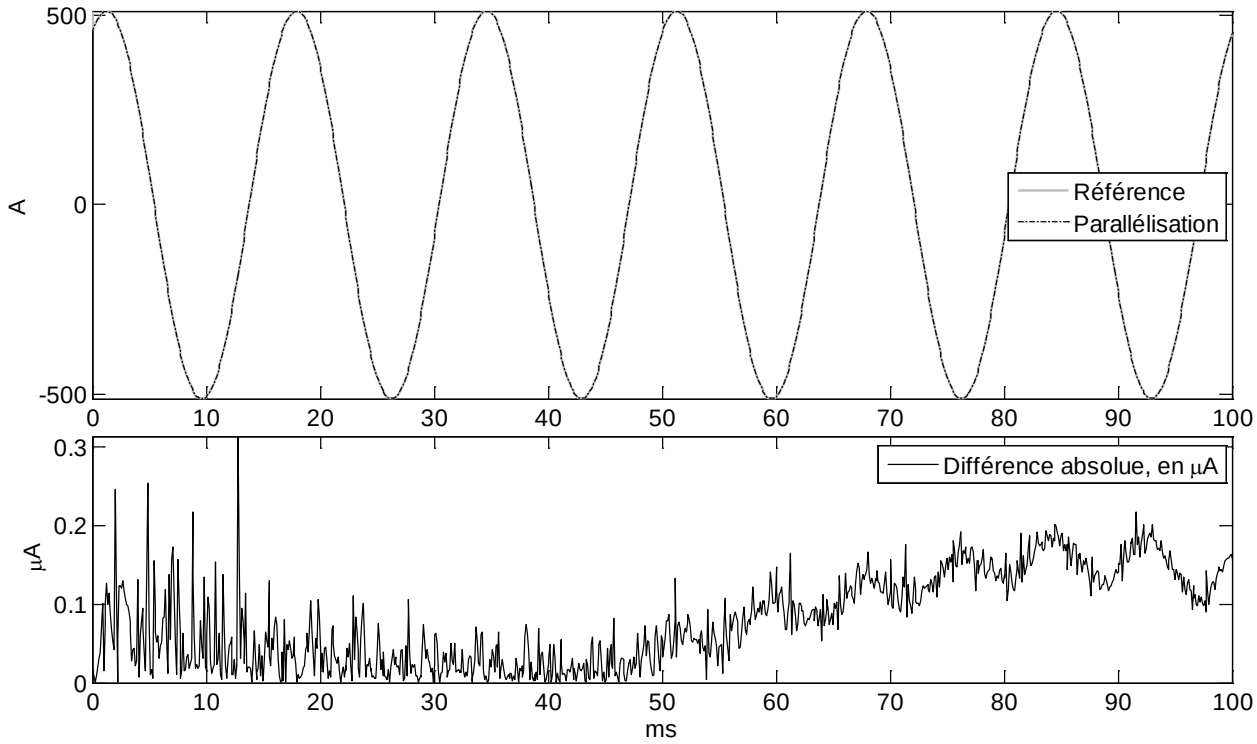

Figure 4-13 Réseau de validation « Kundur », courant machine *SM3*

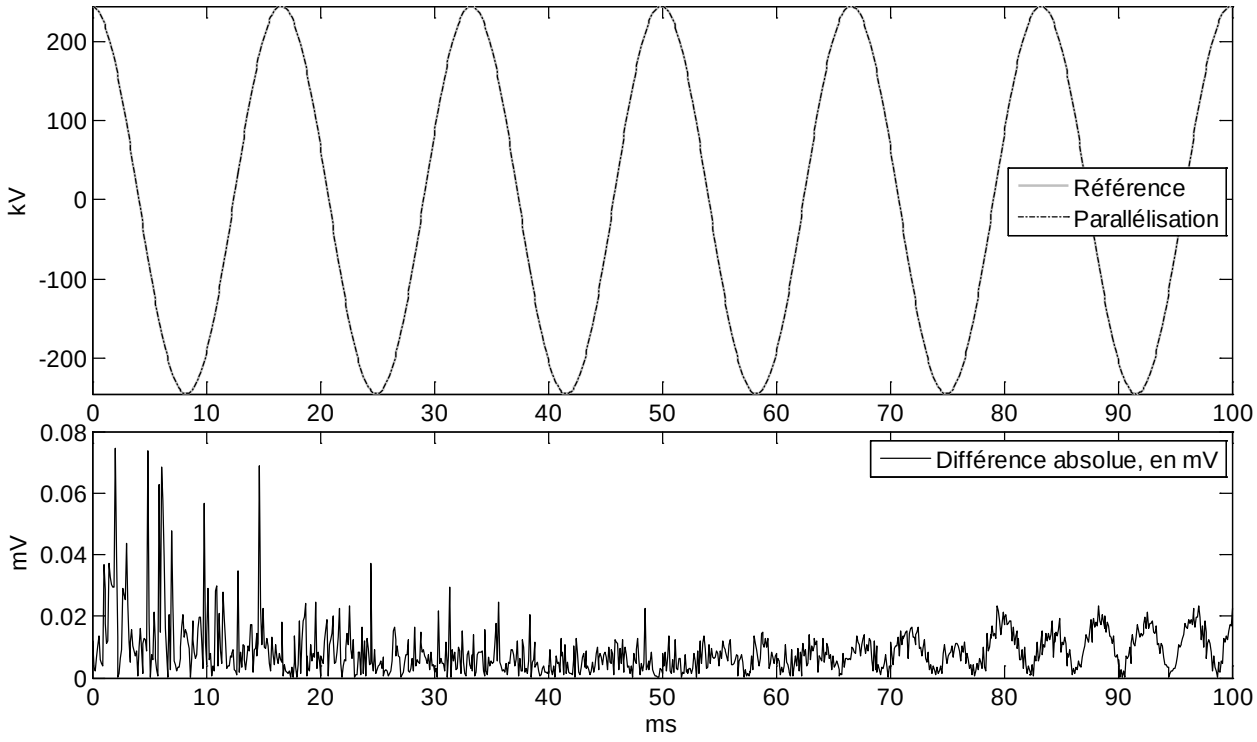

Figure 4-14 Réseau de validation « Kundur », tension machine *SM3*

Le but de l'utilisation d'un réseau de validation comme le « Kundur » est de vérifier le comportement de l'outil en régime permanent et l'initialisation de la simulation dans le domaine du temps à partir du régime initial. En effet, avec l'introduction de modèles de machines synchrones contrôlées, une mauvaise initialisation des machines entraînerait des conséquences sur l'ensemble de la simulation.

Or, comme on le constate sur les figures précédentes, la tension et le courant sont tous deux presque parfaitement initialisés avec l'utilisation de l'outil de parallélisation. En effet, l'erreur sur les signaux atteint au maximum 3×10^{-7} A et 8×10^{-5} V, pour une erreur relative de 6×10^{-8} % et 4×10^{-8} % respectivement. Ainsi, la méthodologie employée pour initialiser les sous-réseaux, décrite à la section 3.3.4, est confirmée par cet exemple.

Pour résumé l'ensemble des essais de validation, on constate que l'erreur relative, sans événement particulier affectant le réseau, est de l'ordre de 10^{-8} %. Si le réseau est soumis à un défaut franc à une extrémité d'une ligne servant de point de découplage, l'erreur peut atteindre 10^{-5} %. Or, les données de réseau utilisées en simulation dans l'industrie des réseaux d'électricité n'atteignent pas cette précision. On affirme que l'outil et ses principales caractéristiques sont validés par ces trois cas de simulation.

4.2 Résultats de performances et analyses préliminaires

Les trois réseaux de tests utilisés pour mesurer la performance de l'outil sont présentés dans cette section. La configuration des réseaux ainsi que leurs variantes sont montrées, puis toutes les configurations de découpage testées sont détaillées avec les résultats pour chacune d'entre elles. Toutes les simulations sont réalisées avec le mode de co-simulation #2 (esclaves simulés en parallèle) et l'optimisation de compilation /O2 décrite à la section 3.3.5.

4.2.1 Réseaux IEEE-9 barres

Le premier réseau d'essai de performance est constitué à partir d'une concaténation de quatre réseaux IEEE-9 barres [41]. Il est important de noter qu'il s'agit ici d'une configuration fictive et le réseau ne correspond pas à un cas pratique.

Étant donné la division naturelle du réseau en quatre zones, seule la performance du cas de la Figure 4-15 est évaluée.

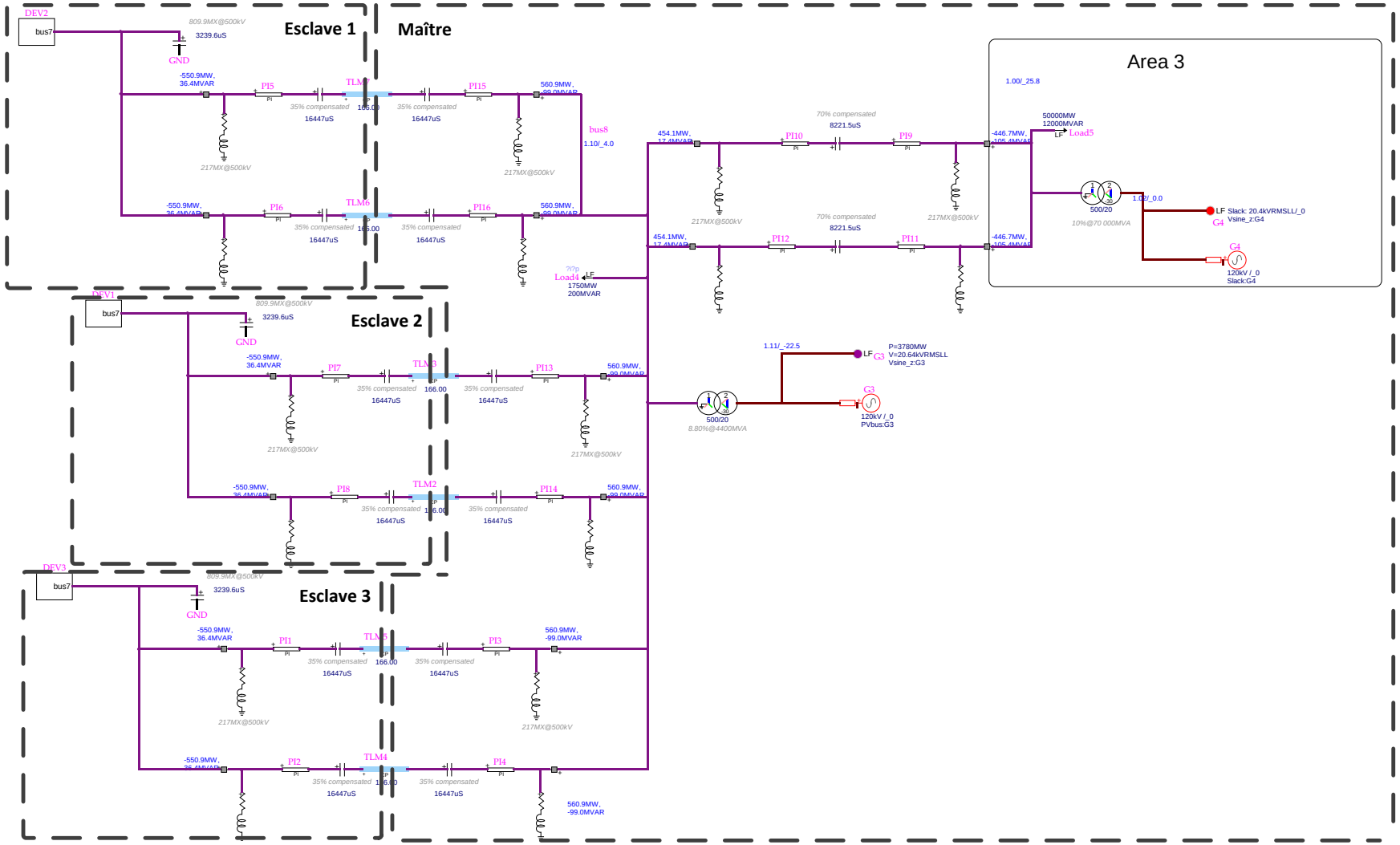

Figure 4-15 Réseau IEEE-9 barres augmenté

Plusieurs processus du système d'exploitation, exécutés en arrière-plan de manière transparente à l'utilisateur, peuvent légèrement ralentir les simulations de manière aléatoire. Ainsi, cinq essais sont réalisés pour chaque configuration, en plus du temps de « Référence » qui correspond au cas où l'ensemble du réseau est simulé sur un seul cœur (sans outil de parallélisation). Le temps retenu est le temps médian des essais, indiqué en gras. L'accélération est calculée avec cet essai et utilisant le temps le plus important des sous-réseaux, puisqu'il correspond au temps réel total de la simulation. Chacune de ces mesures du temps total a été validée avec un chronomètre externe. Des différences minimes ont été observées puisque la mesure externe est dépendante de l'affichage graphique pour déterminer le début et la fin des simulations. Or, cet affichage entraîne un délai supplémentaire qui n'est pas pris en compte par la mesure de temps faite par EMTP. Finalement, des informations temporelles détaillées sont fournies pour le cas médian par rapport à celles du cas de référence.

Pour le réseau IEEE-9 barres augmenté, un seul temps sera donné pour tous les esclaves étant donné que, ces derniers étant parfaitement symétriques, les temps de simulation de chacun d'entre eux sont pratiquement identiques. La taille du système d'équation complet est 1 138 avec 11 600 éléments non-nuls dans la matrice creuse, et le circuit comporte 345 nœuds. Le pas de temps utilisé est de 10 µs. Les longues lignes sont représentées par des modèles CP et les lignes courtes par des sections PI. Le réseau est constitué de sources de tension avec impédances et comprend des contraintes de flux de puissance. Les lignes sont compensées shunt et série et les charges sont des impédances RL présentes dans le calcul du flux de puissance. La saturation des transformateurs n'est pas modélisée.

Tableau 4-1 Réseau IEEE-9 barres augmenté, résultats de performance

Essai	**Temps « maître » (s)**	**Temps « esclaves » (s)**
Référence	**29,9 s**	
1	42,8	24,6
2	**46,2**	**28,5**
3	43,2	25,1
4	48,5	29,8
5	50,7	31,0

On remarque tout d'abord que même si les quatre instances ont terminé la simulation en même temps, le temps que l'UCT a mesuré pour chacune de ces instances n'est pas le même. Ceci est

dû au choix de conception d'opter pour les sémaphores au lieu du verrouillage total de l'UCT (voir section 3.1). En effet, lorsqu'une instance a terminé les calculs de son pas de temps, elle reste en attente qu'un sémaphore (#2 ou #3 selon) se libère. Pendant cette attente, l'UCT associée à l'instance en question ne reste pas active, mais entre en veille ou consacre sa puissance de calcul à d'autres processus. Ce temps d'attente n'est pas comptabilisé dans le temps mesuré par l'UCT pour la simulation EMTP, et c'est pourquoi l'instance la moins chargée montre un temps de simulation plus court.

Puis, on voit que même en divisant ce réseau en quatre sous-réseaux, l'utilisation de l'outil de parallélisation n'apporte aucun gain de performance. Le Tableau 4-2 présente la comparaison des temps détaillés de chaque fonction entre le cas de référence et la simulation avec l'outil de parallélisation (temps médian des cinq essais). On constate que le gain réalisé sur la charge de calcul (15,1 s versus 8,39 s) n'est tout simplement pas assez important par rapport au délai de communication entre les instances (8,52 s versus 29,7 s). Le nombre moyen d'itérations par pas de temps maximal parmi les instances est de 1,02 pour l'instance maître.

Tableau 4-2 Réseau IEEE-9 barres augmenté, comparaison des temps détaillés

Fonctions	**Temps « Référence » (s)**	**Temps « Avec outil » (s)**
Mise à jour du vecteur **b**	2,36	2,34
Solution **Ax = b**	15,1	8,39
Solution du système de contrôle	0.06	0,20
Mise à jour de l'historique (communication)	8,52	29,7
Solution du domaine du temps	29,8	46,0

4.2.2 Réseau IEEE-39 barres

Le deuxième réseau utilisé est un cas de test développé par l'IEEE, communément appelé IEEE-39 barres [42]. Quatre configurations seront étudiées pour ce réseau. La première contient seulement des machines synchrones, alors qu'on a introduit deux parcs éoliens dans les trois dernières. Tous ces réseaux simulent un défaut triphasé à la barre B3 suivi d'un déclenchement de la ligne *bus03_04* avec un pas de temps de 50 µs. Les longues lignes de transmission du réseau IEEE-39 barres sont des lignes CP utilisant le modèle à paramètre distribuées. Les lignes

courtes sont représentées par des sections PI. Tous les groupes de production et les charges sont modélisés avec trois unités indépendantes avec données de magnétisation. Les machines synchrones utilisent l'approche de modélisation dq0, avec une seule masse et régulation de tension et de puissance (AVR/GOV). Toutes les charges incluent la dépendance en tension et en fréquence [43]. Ce réseau permet aussi d'étudier des phénomènes électromécaniques.

La première configuration est présentée à la Figure 4-16. À la suite des résultats acquis avec le réseau IEEE-9 barres augmenté, le réseau n'est découplé qu'en deux instances afin de charger davantage chaque UCT. Trois bus de co-simulation sont utilisés. Le circuit original comporte 478 nœuds, une matrice d'équation d'une taille de 730x730 comptant 3 082 éléments non-nuls, 1 333 appareils de puissance et 5 443 blocs de contrôle.

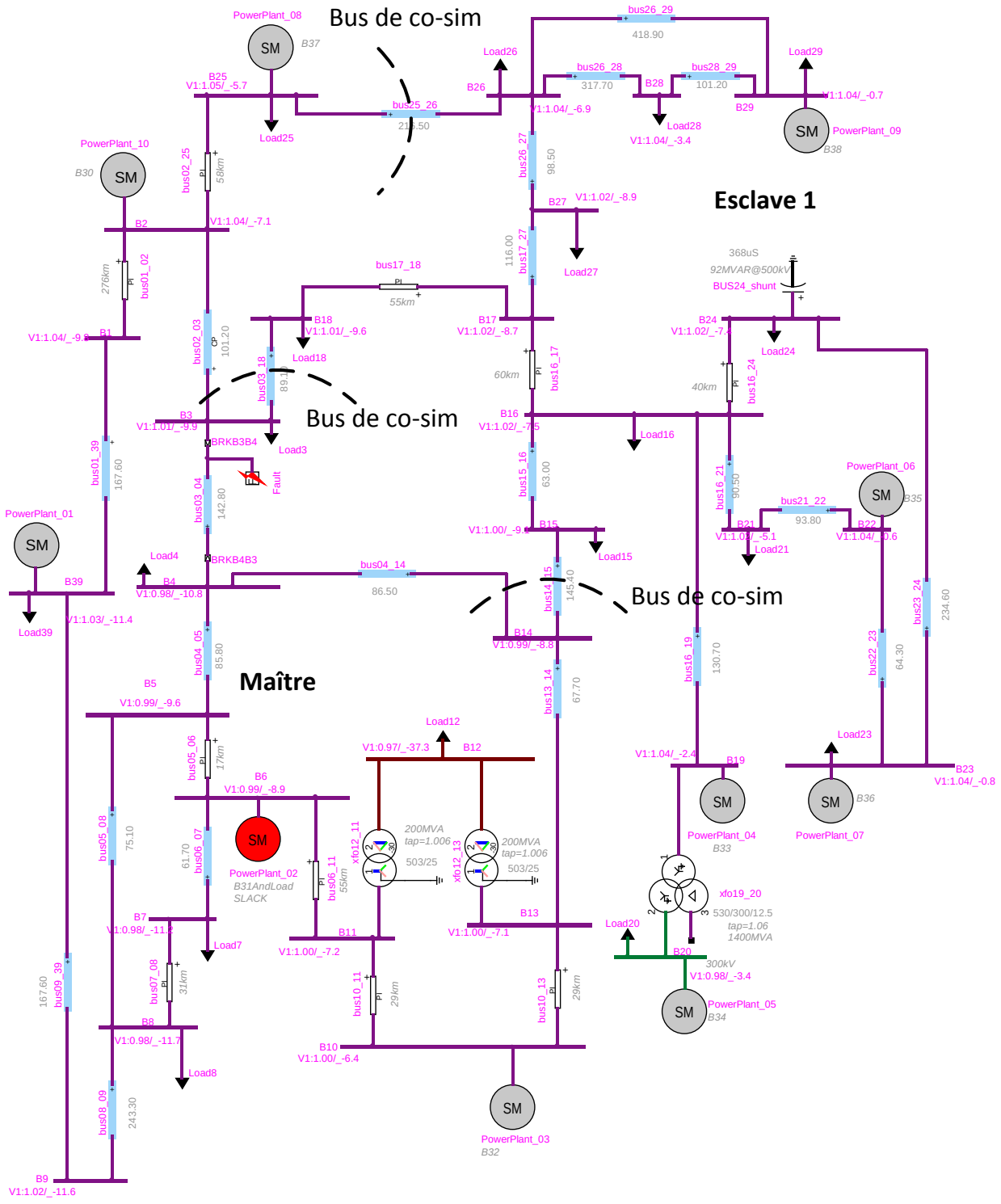

Figure 4-16 Réseau IEEE-39 barres, configuration #1

Tableau 4-3 Réseau IEEE-39 barres, configuration #1, résultats de performance

Essai	**Temps « maître » (s)**	**Temps « esclaves » (s)**	**Temps réel (s)**
Référence	**210 s**		
1	135	151	153
2	162	178	180
3	127	145	147
4	**140**	**161**	**162**
5	160	182	182

On constate qu'avec ce cas, on obtient un gain de performance de **23 %** avec l'outil de co-simulation par rapport au cas de référence. Le nombre moyen d'itérations par pas de temps est de 1,34. On présente les temps détaillés de l'essai médian au Tableau 4-4.

Tableau 4-4 Réseau IEEE-39 barres, configuration #1, comparaison des temps détaillés

Fonctions	**Temps « Référence » (s)**	**Temps « Avec outil » (s)**
Mise à jour du vecteur **b**	1,54	1,65
Solution **Ax = b**	20,5	20,3
Solution du système de contrôle	170	122
Mise à jour de l'historique (communication)	5,05	6,31
Solution du domaine du temps	209	161

Avec un seul réseau esclave, la perte de temps causée par le partage des données (5,05 s versus 6,31 s) est beaucoup plus faible que dans l'exemple précédent qui comportait trois esclaves. Ici, le gain de temps au niveau de la solution du système de puissance est négligeable, alors qu'il est observé plutôt au niveau des systèmes de contrôle. En effet, contrairement au réseau IEEE-9 barres augmenté qui ne comporte que des sources de tension, toutes les machines du réseau IEEE-39 barres sont dotées de systèmes de contrôle de tension et de puissance qui représentent une charge de calcul importante à chaque pas de temps.

On poursuit les tests de performance en utilisant une variante du réseau IEEE-39 barres, qui remplace deux machines synchrones des barres B2 et B25 par deux sous-réseaux de parcs éoliens terrestres sur les barres B1, B2 et B25. Le premier, nommé *Onshore_WP1*, contient deux parcs éoliens agrégés de Type-III et deux de Type-IV générant 255 MW. Le second, *Onshore_WP2*,

comporte sept parcs agrégés, un seul de Type-IV et les autres de Type-III. Le réseau de base compte 2 336 appareils de puissance et 24 364 blocs de contrôle. Le système d'équation du réseau complet est d'une taille de 2 588 par 2 588, et la matrice contient 9 288 éléments non-nuls. Afin de tester les performances de l'outil avec ce réseau modifié, trois configurations de découpage seront expérimentées. Le nombre moyen d'itérations par pas de temps pour les trois configurations testées pour cette variante augmente à 3 étant donné la présence des modèles de parcs éoliens. Le pas de temps utilisé pour les trois configurations est 50 µs.

La première configuration correspond à celle illustrée à la Figure 4-17. Cette dernière porte également le nom de « configuration #1 » puisque le découpage est exactement le même que pour le réseau IEEE-39 barres précédent. Le réseau est découpé en une instance maître et une instance esclave par trois bus de co-simulation.

La seconde configuration est testée à la suite des résultats obtenus avec la configuration #1. En effet, les éoliennes, qui sont toutes dans la même instance avec la configuration #1, sont réparties en deux instances selon les Figure 4-18 et Figure 4-20. Le découplage est réalisé uniquement par deux bus de co-simulation.

Puis, afin d'obtenir un découplage plus efficace, une troisième configuration pour le réseau IEEE-39 barres avec éoliennes est vérifiée. Cette dernière regroupe l'ensemble des éoliennes du parc *Onshore_WP2*, deux éoliennes du parc *Onshore_WP1* et une machine dans l'instance « maître ». Les cinq éoliennes restantes du parc Onshore_WP2 sont dans l'instance esclave #1 et le reste du réseau est dans l'instance esclave #2. Cinq bus de co-simulation découplent les trois instances. C'est cette troisième configuration, présentée à la Figure 4-19 et à la Figure 4-20, qui procure les meilleurs résultats de performance. Le découpage à l'intérieur du parc éolien Onshore_WP2 est le même pour la configuration #2 et la configuration #3, et il est illustré à la Figure 4-20.

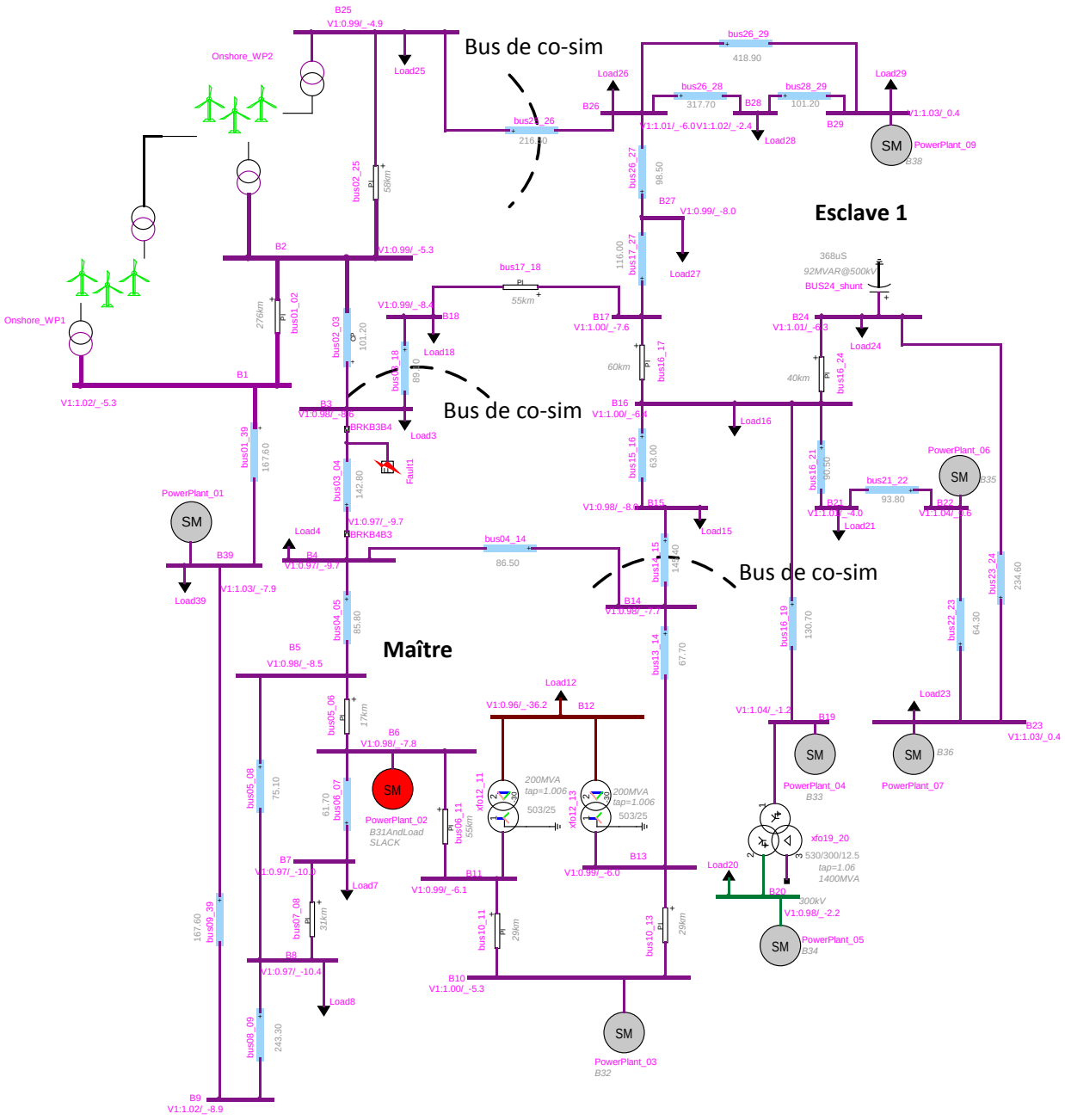

Figure 4-17 Réseau IEEE-39 barres avec éoliennes, configuration #1

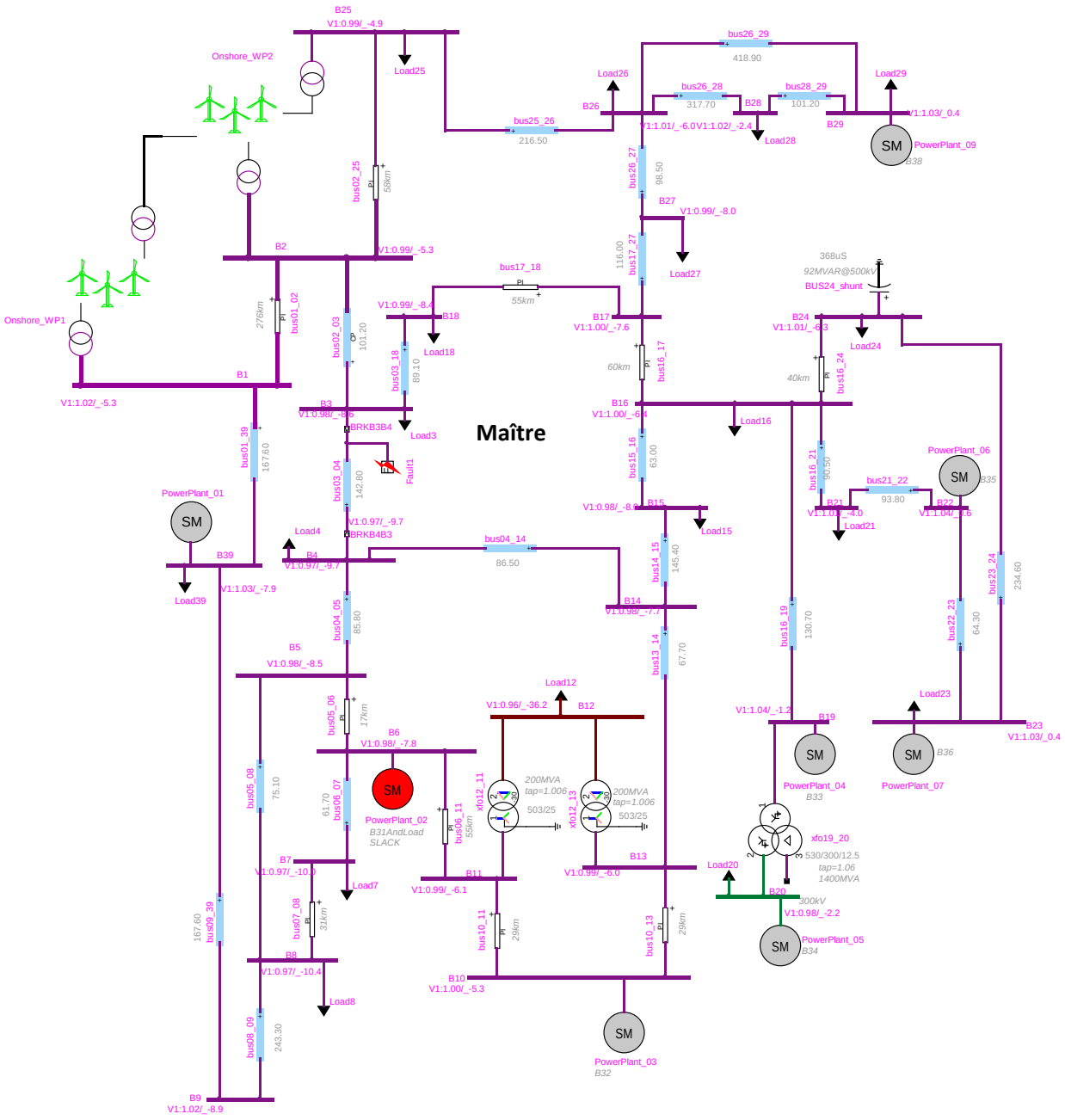

Figure 4-18 Réseau IEEE-39 barres avec éoliennes, configuration #2

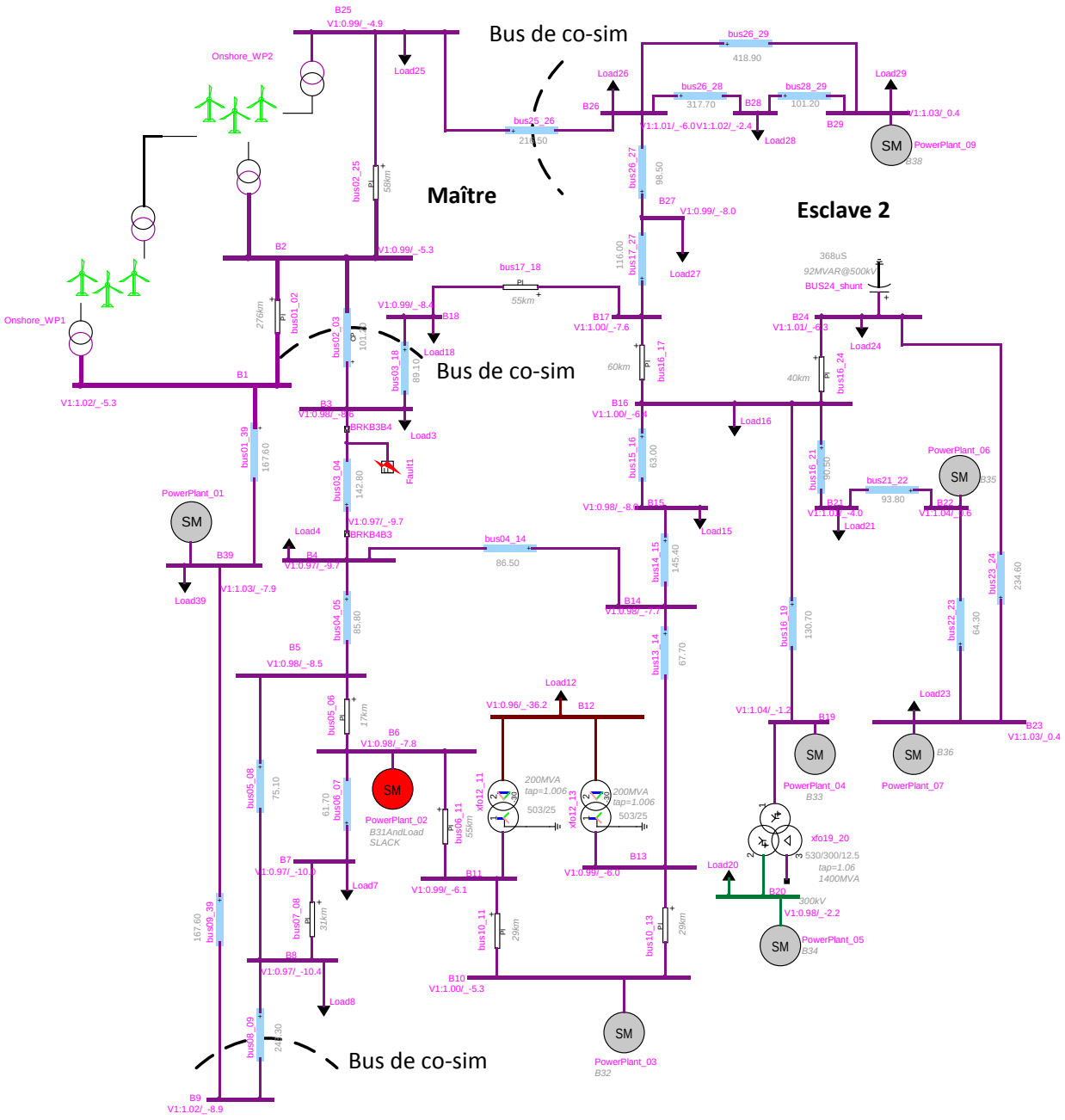

Figure 4-19 Réseau IEEE-39 barres avec éoliennes, configuration #3

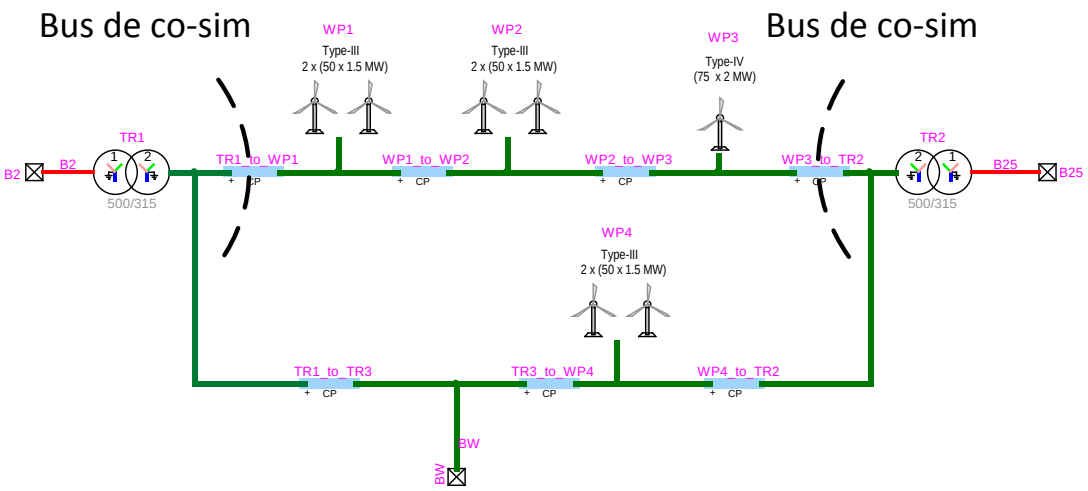

Figure 4-20 Sous-réseau du parc éolien Onshore_WP2, configurations #2 et #3

Tableau 4-5 Réseau IEEE-39 barres avec éoliennes, configuration #1, résultats de performance

Essai	**Temps « maître » (s)**	**Temps « esclave » (s)**
Référence	**641 s**	
1	**580**	**53,6**
2	577	53,4
3	578	50,0
4	586	55,7
5	580	52,1

Tableau 4-6 Réseau IEEE-39 barres avec éoliennes, configuration #1, comparaison des temps détaillés

Fonctions	**Temps « Référence » (s)**	**Temps « Avec outil » (s)**
Mise à jour du vecteur **b**	1,30	1,42
Solution **Ax = b**	38,5	41,7
Solution du système de contrôle	559	495
Mise à jour de l'historique (communication)	5,87	6,88
Solution du domaine du temps	626	566

Tout d'abord, on constate un léger gain de performance de **9,5 %** avec la configuration #1. On peut rapidement évaluer que le découpage du réseau n'est pas optimal par la différence majeure entre le temps requis par l'instance maître par rapport à l'instance esclave (580 s versus 53,6 s). L'essai de la deuxième configuration de découpage est donc réalisé selon la disposition des lignes CP disponibles, soit avec deux instances et deux bus de co-simulations. Les résultats de performance obtenus avec la configuration #2 (Figure 4-18 et Figure 4-20) sont présentés au Tableau 4-7 et au Tableau 4-8.

Tableau 4-7 Réseau IEEE-39 barres avec éoliennes, configuration #2, résultats de performance

Essai	**Temps « maître » (s)**	**Temps « esclave » (s)**
Référence	**641 s**	
1	417	236
2	426	231
3	422	236
4	419	231
5	**420**	**233**

Tableau 4-8 Réseau IEEE-39 barres avec éoliennes, configuration #2, comparaison des temps détaillés

Fonctions	**Temps « Référence » (s)**	**Temps « Avec outil » (s)**
Mise à jour du vecteur **b**	1,30	1,36
Solution **Ax = b**	38,5	28,4
Solution du système de contrôle	559	362
Mise à jour de l'historique (communication)	5,87	6,68
Solution du domaine du temps	626	413

Avec la configuration #2, on constate une meilleure répartition de la charge de calcul entre les deux instances (420 s versus 233 s). Le gain de performance obtenu atteint les **34 %**. L'importance du temps nécessaire à la solution du système de contrôle rend la parallélisation très efficace. En effet, les modèles d'éoliennes sont très lourds à calculer et la répartition de ceux-ci entre les deux instances entraîne des gains importants au niveau du système de contrôle (559 s versus 362 s). Or, la répartition n'est toujours pas optimale puisque l'instance maître est trop chargée. Le découpage du réseau est donc réorganisé selon la configuration #3 (Figure 4-19) avec

trois instances et 6 bus de co-simulation. On obtient les résultats montrés au Tableau 4-9 et au Tableau 4-10.

Tableau 4-9 Réseau IEEE-39 barres avec éoliennes, configuration #3, résultats de performance

Essai	**Temps « maître » (s)**	**Temps « esclave 1 » (s)**	**Temps « esclave 2 » (s)**
Référence	**641 s**		
1	361	283	217
2	363	279	203
3	349	288	205
4	**357**	**286**	**207**
5	350	290	212

Tableau 4-10 Réseau IEEE-39 barres avec éoliennes, configuration #3, comparaison des temps détaillés

Fonctions	**Temps « Référence » (s)**	**Temps « Avec outil » (s)**
Mise à jour du vecteur **b**	1,30	0,858
Solution **Ax = b**	38,5	34,4
Solution du système de contrôle	559	300
Mise à jour de l'historique (communication)	5,87	7,86
Solution du domaine du temps	626	352

On observe une amélioration de la répartition de la charge de calcul en utilisant trois sous-réseaux. On obtient un gain de performance de **44 %**. Le temps requis pour la communication des données augmente étant donné l'utilisation de trois instances au lieu de deux dans les configurations précédentes. Or, cette perte est largement compensée par la réduction du temps de calcul du système de contrôle (559 s versus 300 s). Étant donné la topologie de ce réseau, on considère ce découpage comme étant optimal et les gains de performance, très satisfaisants.

4.2.3 Réseau de transmission réel

Le dernier réseau d'essais de performance est un réseau de transmission de réel, employé comme cas de test par les développeurs du logiciel EMTP [44]. Ce réseau est important pour l'évaluation de la performance puisqu'il correspond à une topologie de réseau réelle, un aspect qui est parfois

absent des études de parallélisation des logiciels. C'est toutefois un aspect crucial pour une évaluation complète de l'amélioration des performances.

Trois variantes seront étudiées, correspondant à trois degrés de pénétration de production éolienne. La première variante (Figure 4-21) est formée d'un réseau sans parc éolien. La production est entièrement assurée par des machines synchrones contrôlées en tension et en puissance. La seconde variante (Figure 4-22) comporte un parc éolien ajouté par rapport à la variante sans production éolienne. Finalement, la troisième variante (Figure 4-23) présente un réseau où deux sous-réseaux éoliens ont été ajoutés à la production. Les réseaux collecteurs des parcs éoliens ne sont pas modélisés par des lignes CP, nous empêchant d'utiliser le découplage à l'intérieur d'un même parc. Le réseau simule un défaut triphasé sur la barre ADAPA suivi de l'élimination du défaut par ouverture de ligne.

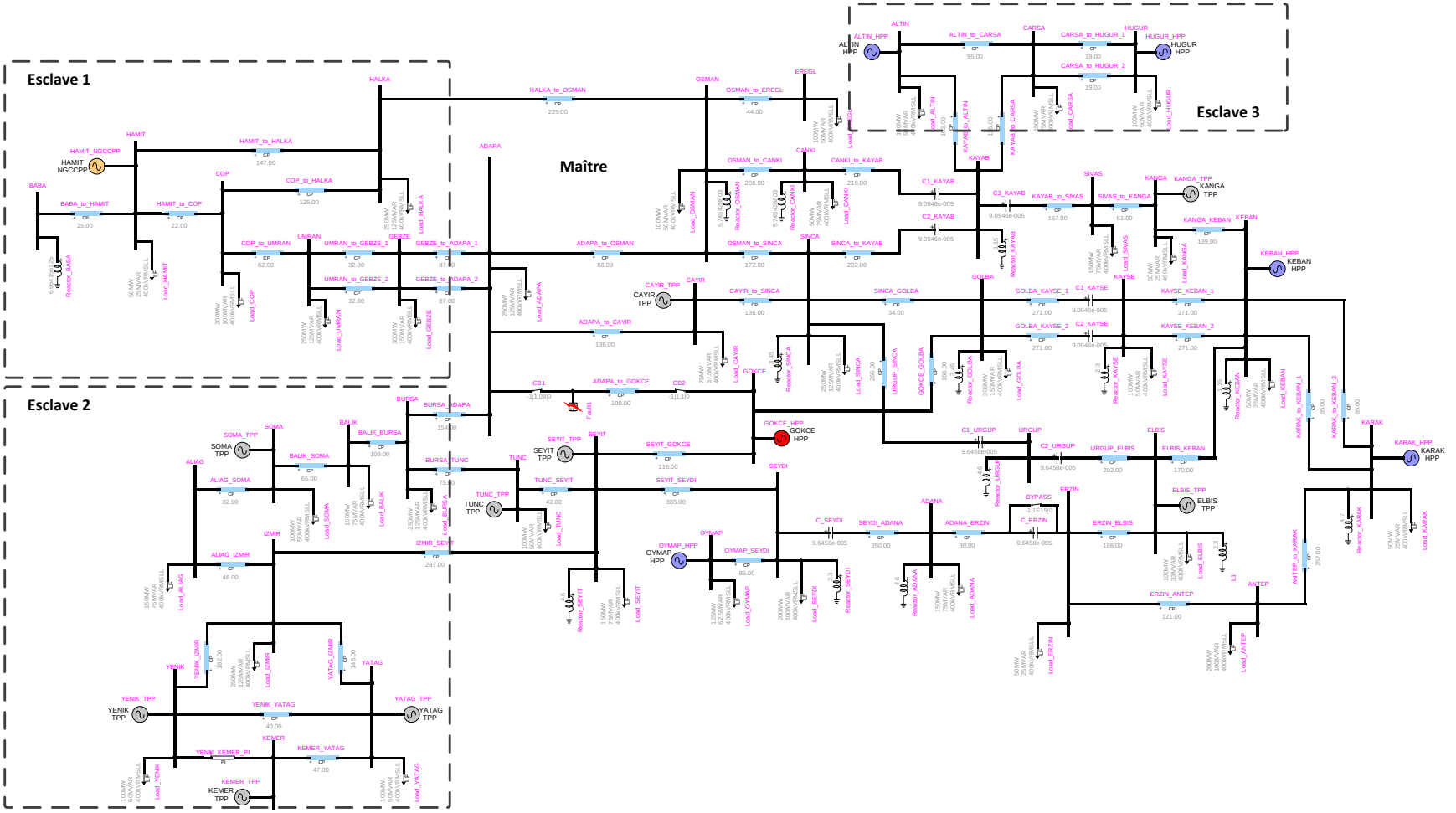

Figure 4-21 Réseau de transmission réel, sans parc éolien

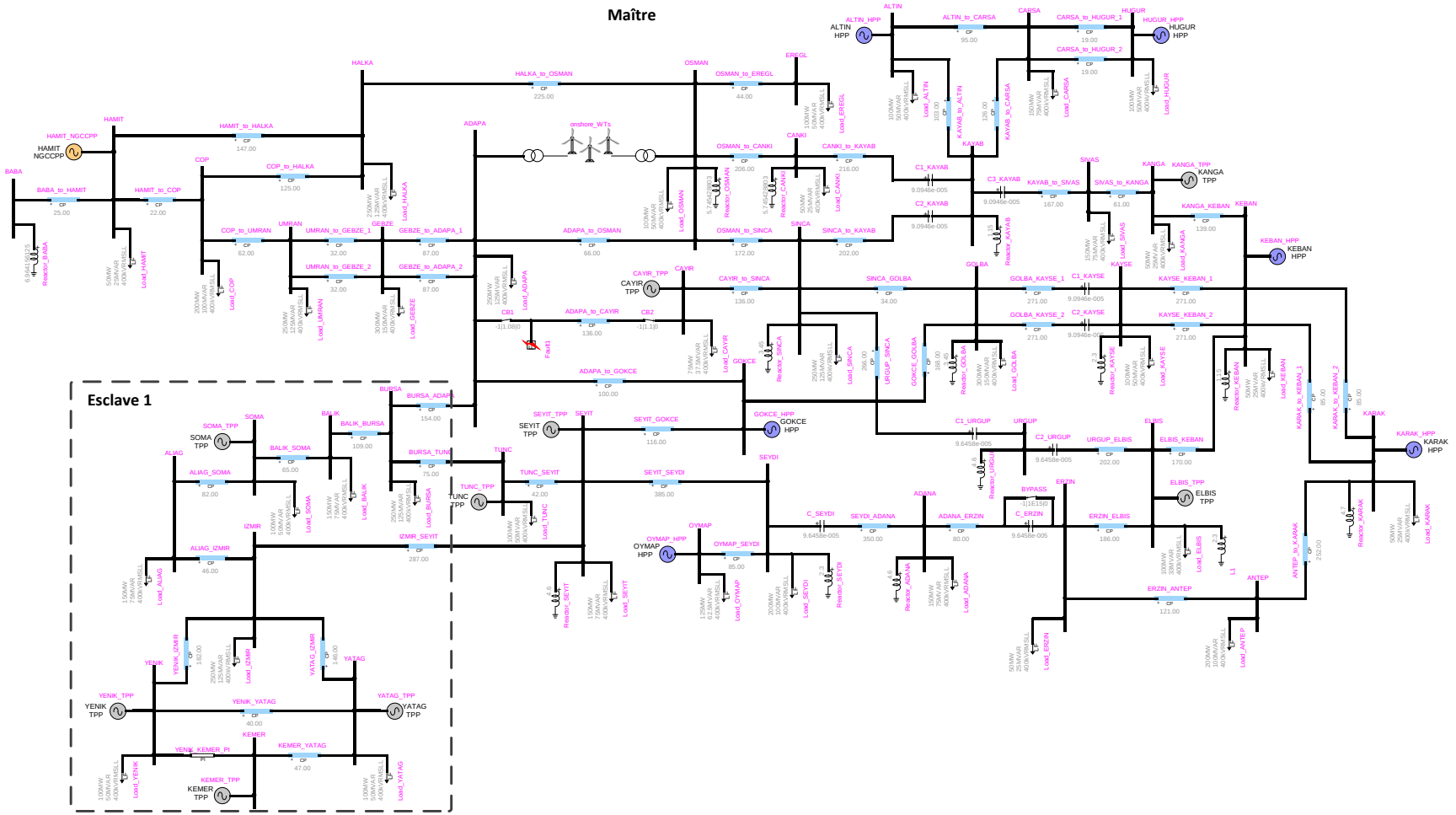

Figure 4-22 Réseau de transmission réel, avec 1 parc éolien

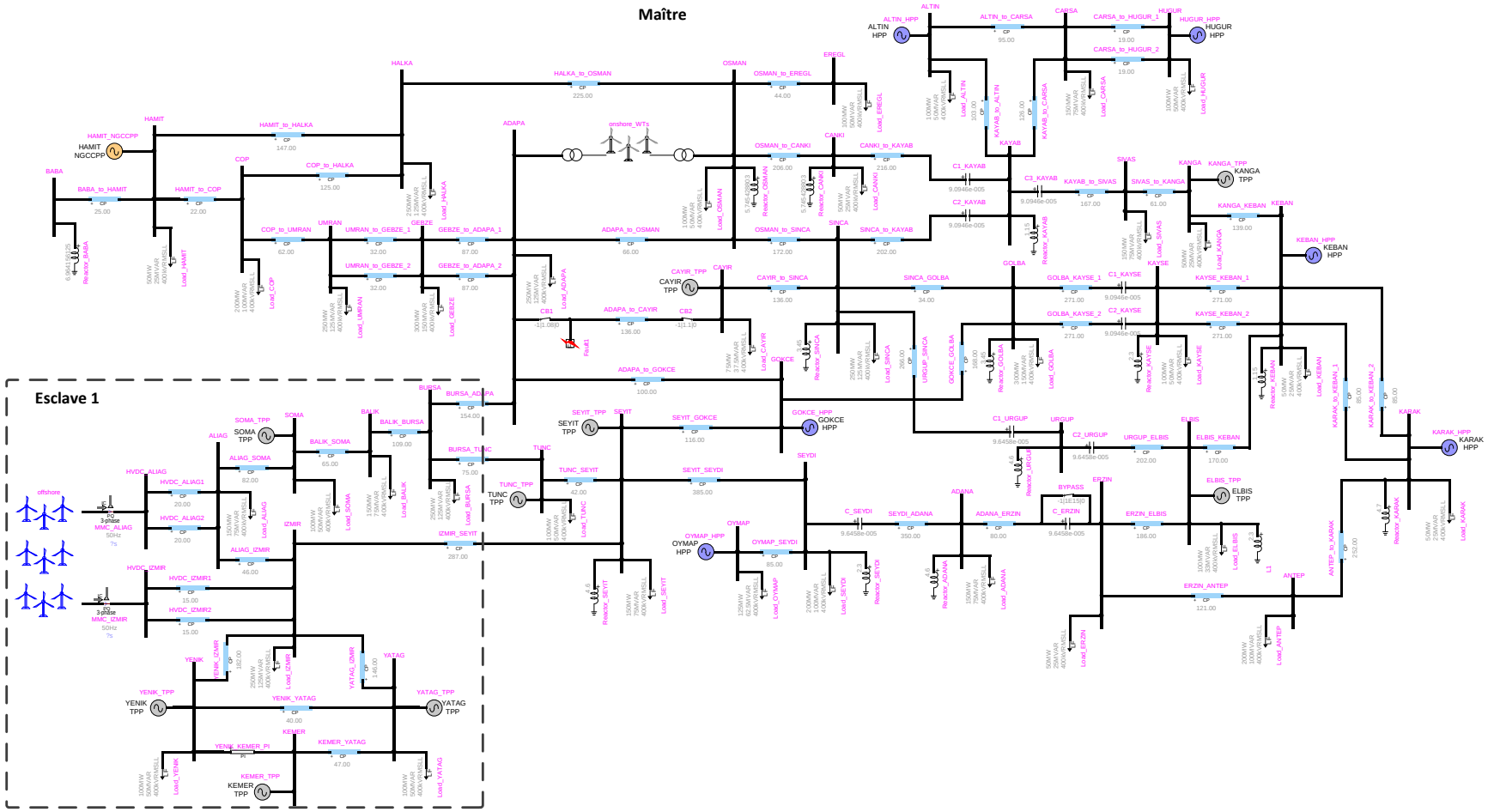

Figure 4-23 Réseau de transmission réel, avec 2 parcs éoliens

La variante #1 du réseau réel est composée de 35 machines synchrones avec excitatrice, gouverneur et données de saturation. Le nombre de nœuds s'élève à 469, le système d'équation est d'une taille de 691 lignes avec 3535 éléments non-nuls. Les charges sont maintenant désormais représentées comme étant constantes dans le temps, contrairement à l'exemple précédent. Le pas de temps de simulation pour toutes les variantes est de 50 μs. Le réseau est découplé avec quatre instances (un maître et trois esclaves) par 8 bus de co-simulation.

Tableau 4-11 Réseau réel sans parc éolien, résultats de performance

Essai	**Temps « maître » (s)**	**Temps « esclave 1 » (s)**	**Temps « esclave 2 » (s)**	**Temps « esclave 3 » (s)**
Référence	**59,1 s**			
1	50,9	28,1	33,0	21,7
2	**52,3**	**30,4**	**37,4**	**24,1**
3	52,0	29,1	38,8	23,3
4	51,9	29,9	38,2	21,0
5	54,8	32,7	44,4	25,9

Tableau 4-12 Réseau réel sans parc éolien, comparaison des temps détaillés

Fonctions	**Temps « Référence » (s)**	**Temps « Avec outil » (s)**
Mise à jour du vecteur **b**	2,22	2,68
Solution **Ax = b**	10,7	7,69
Solution du système de contrôle	30,2	19,2
Mise à jour de l'historique (communication)	7,32	13,0
Solution du domaine du temps	58,8	51,4

Pour la première variante, on mesure un faible gain de performance de **12 %** par rapport au cas de référence. On explique ce résultat par la faible charge de calcul des quatre UCT impliquées dans le calcul parallèle de ce réseau. En effet, même si les sous-réseaux semblent bien équilibrés par des temps de simulation comparables, le gain obtenu sur la solution du système de contrôle (30,2 s versus 19,2 s) est à peine suffisant pour compenser la perte entraînée par les délais de communication (7,32 s versus 13,0 s). Le nombre d'itérations moyen par pas de temps est de 1.

Pour l'évaluation de la performance des variantes #2 et #3, on réduit à deux le nombre d'instances (un maître et un esclave) et 3 bus de co-simulation. Les résultats de la variante #2 sont présentés au Tableau 4-13 et au Tableau 4-14. Un sous-réseau éolien terrestre de 322,5 MW composé de trois parcs d'éoliennes de Type-III et deux parcs Type-IV est ajouté entre les barres ADAPA et OSMAN, et deux machines synchrones sont retirées. Le circuit compte 964 nœuds, un système de 1 542 équations avec 6364 éléments non-nuls dans la matrice creuse, et un total de 1 284 appareils et 10 420 blocs de contrôle.

Tableau 4-13 Réseau réel avec 1 parc éolien, résultats de performance

Essai	**Temps « maître » (s)**	**Temps « esclave » (s)**
Référence	**77,2 s**	
1	**74,8**	**4,54**
2	72,8	3,39
3	76,3	4,22
4	73,0	3,97
5	77,9	5,65

Tableau 4-14 Réseau réel avec 1 parc éolien, comparaison des temps détaillés

Fonctions	**Temps « Référence » (s)**	**Temps « Avec outil » (s)**
Mise à jour du vecteur **b**	0,406	0,359
Solution **Ax = b**	5,60	5,01
Solution du système de contrôle	61,3	59,7
Mise à jour de l'historique (communication)	1,15	1,50
Solution du domaine du temps	73,0	70,9

On note ici un très faible gain de performance d'à peine **3 %**. La performance de l'outil pour cette variante en particulier est expliquée par le débalancement marqué de la charge de calcul entre l'instance maître et l'instance esclave. La première contient cinq éoliennes et douze machines synchrones alors que la seconde ne comprend que quatre machines synchrones. Ce découpage est identique à celui de la variante à deux parcs afin de comparer l'impact de l'ajout d'un parc éolien sur la performance. En effet, comme il a également été constaté avec le réseau IEEE-39 barres avec éoliennes à la section 4.2.2, le travail requis à chaque pas de temps pour

calculer le modèle d'éoliennes utilisé ici est grandement supérieur à celui demandé par la machine synchrone contrôlée. Or, étant donné que le réseau du parc éolien n'est pas modélisé avec des lignes CP, ce dernier ne peut être découplé par l'outil de parallélisation. Cette variante particulière constitue donc un exemple de cas réel où la parallélisation par découplage aux lignes de transmission s'applique difficilement. Il incombera à l'utilisateur de cet outil d'évaluer les réseaux où la méthode proposée dans le présent mémoire peut être utilisée de manière efficace, et où une autre méthode d'accélération du logiciel serait à favoriser. Aussi, le nombre moyen d'itérations augmente à 3 étant donné la présence d'éoliennes.

La variante #3 avec deux sous-réseaux éoliens présente une configuration naturellement favorable au découplage en deux sous-réseaux tel que défini à la Figure 4-23. Le second parc en mer de 1 057 MW est relié par un terminal multiple HVDC avec convertisseurs MMC comprenant 400 modules séries par branche [45], [46] (Figure 4-24). Le circuit comporte 1 480 nœuds, 2 425 équations avec 9 359 éléments non-nuls, 1 997 blocs de puissance et 17 931 blocs de contrôles. Les résultats obtenus en utilisant l'outil de parallélisation sont donnés au Tableau 4-15 et au Tableau 4-16.

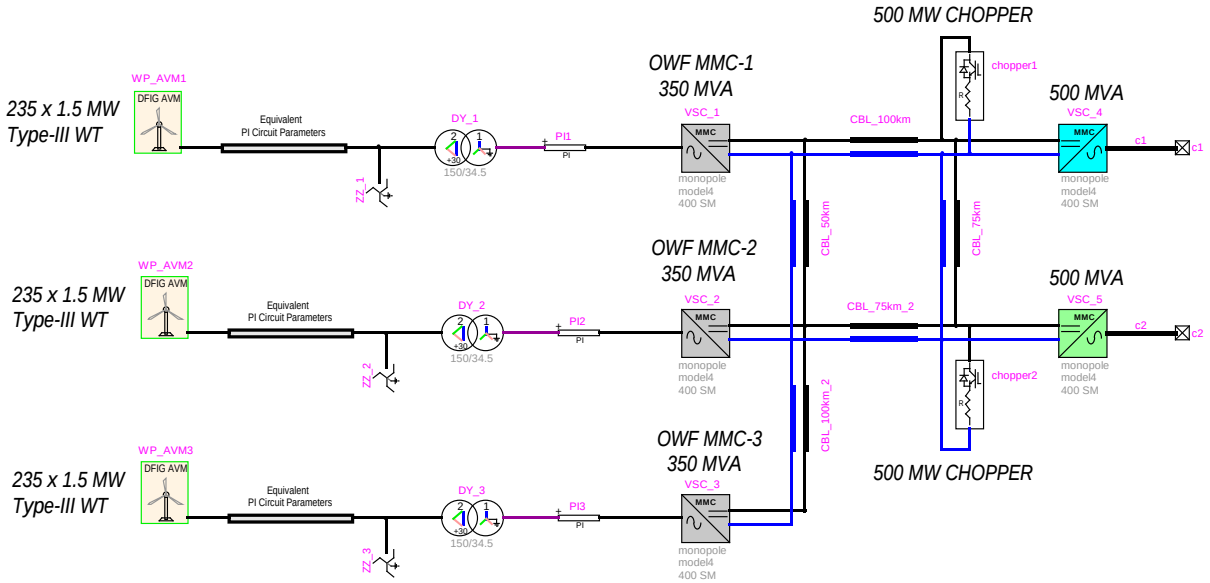

Figure 4-24 Parcs éoliens en mer avec transmission par HVDC-MMC

Tableau 4-15 Réseau réel avec 2 parcs éoliens, résultats de performance

Essai	**Temps « maître » (s)**	**Temps « esclave » (s)**
Référence	**164 s**	
1	89,7	81,7
2	83,4	78,2
3	**86,5**	**80,1**
4	88,9	80,0
5	83,0	82,2

Tableau 4-16 Réseau réel avec 2 parcs éoliens, comparaison des temps détaillés

Fonctions	**Temps « Référence » (s)**	**Temps « Avec outil » (s)**
Mise à jour du vecteur **b**	0,406	0,374
Solution **Ax = b**	8,30	4,63
Solution du système de contrôle	133	71,3
Mise à jour de l'historique (communication)	2,79	1,65
Solution du domaine du temps	153	81,8

Ce cas présente la meilleure performance de l'outil de parallélisation parmi les réseaux étudiés avec un gain de **47 %**. Ce réseau est séparé en seulement deux instances, réduisant les délais de communication. De plus, avec un parc éolien exigeant un temps de calcul important par instance, le gain réel observé se rapproche du gain maximal théorique de 50 % pour une simulation entièrement parallélisable (équation 2.26). La topologie du réseau se prête parfaitement à l'utilisation de l'outil de parallélisation développé dans le cadre de ce projet. Aussi, le nombre moyen d'itérations par pas de temps demeure à 3.

4.3 Discussion et analyse globale des résultats de performance

En résumé, on peut observer le gain relatif au temps de référence des variantes et configurations de découplage étudiées à la Figure 4-25. On constate que plus un réseau contient de modèles complexes (ex. parcs éoliens, convertisseurs), plus les gains obtenus par la parallélisation seront importants. Ces gains sont d'autant plus considérables que les temps de communication sont relativement faibles par rapport aux temps totaux. Ces améliorations sont également directement

dépendantes d'un partage équitable de la charge de calcul entre les sous-réseaux, qui doit être soigneusement étudié par l'utilisateur, ou pris en charge par un système de découpage automatique qui fait partie des améliorations futures de l'outil.

À plusieurs reprises à la section 4.2, nous avons discuté du délai de temps entraîné par l'utilisation de l'outil de parallélisation. Or, il est difficile, avec les outils à notre disposition pendant ce projet, de quantifier avec précision l'impact du nombre de bus de co-simulation sur l'augmentation du temps de calcul dans la fonction de communication *dll_update_at_t*. En effet, comme on peut le constater en comparant les cinq essais de chaque réseau, le temps de calcul est variable selon des paramètres aléatoires. Cette variation est de l'ordre d'une à plusieurs secondes entre les différents essais, selon le temps total, pour des simulations identiques. Or, dès que le réseau est plus volumineux, le temps de communication entre les instances est de l'ordre de grandeur de cette variation. On peut toutefois faire l'observation qualitative que les réseaux de simulation nécessitant un découplage à plusieurs bus de co-simulation (c'est-à-dire, les réseaux fortement maillés) ont une augmentation importante du temps requis par la fonction *dll_update_at_t*. En effet, on mesure une augmentation sur les réseaux IEEE-9 barres augmenté (six bus de co-simulation) et le réseau réel sans parc éolien (neuf bus de co-simulation) de 21 s et 6 s respectivement, soit 45 % et 12 % du temps total. Cet impact du délai de communication des données dépend donc de la grandeur et de la complexité du réseau. Au contraire, lorsqu'un découpage nécessite moins de bus de co-simulation, on note une augmentation d'une à deux secondes, qui est la plupart du temps négligeable selon le temps de simulation total du réseau.

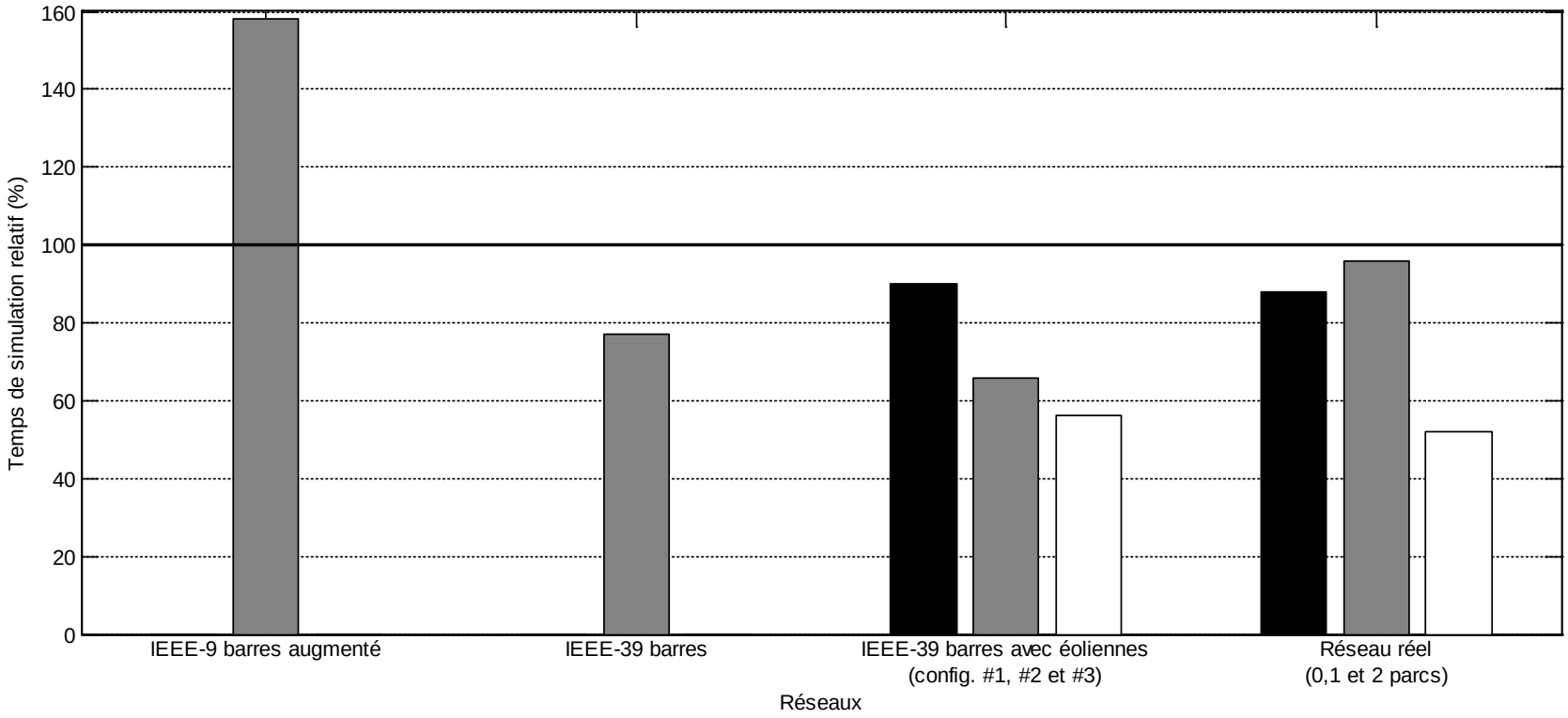

Figure 4-25 Résumé des temps de simulation relatifs aux temps de référence, en %

CONCLUSION

L'objectif général du projet de maîtrise présenté à l'intérieur de ce mémoire était d'accélérer les simulations dans le domaine du temps du logiciel EMTP par l'utilisation de l'architecture multi-cœurs des ordinateurs personnels modernes. Cet objectif a été atteint par l'accomplissement de quatre objectifs spécifiques.

1. **Étudier les approches de programmation parallèle pour des applications de type EMTP.**

Plusieurs méthodes provenant de la littérature sur le sujet ont été détaillées dans le Chapitre 2. L'utilisation ainsi que les avantages et inconvénients de chacune ont été présentés.

2. **Choisir une approche de parallélisation éprouvée et livrable dans le cadre d'un projet de maîtrise.**

À partir des solutions existantes, une stratégie de découplage en sous-réseaux a été adoptée pour sa simplicité et sa rapidité d'implémentation dans le logiciel EMTP. Ce choix est détaillé à la section 3.1

3. **Développer et utiliser des cas de tests pour valider l'approche de parallélisation et analyser les gains qui en résultent.**

Cinq cas de test et leurs variantes ont été utilisés dans le cadre de ce projet et présentés au Chapitre 4.

4. **Proposer une preuve de concept d'une méthode de résolution en parallèle en démontrant les gains en temps de calcul sur des cas pratiques.**

À partir d'un outil existant, plusieurs ajouts et modifications ont été nécessaires afin de rencontrer tous les besoins de conception requis pour simuler l'ensemble des cas de tests proposés. La première modification majeure a été de modifier les signaux transités, les requêtes provenant du solveur et le schéma de synchronisation afin de partager des signaux de puissance et non pas des signaux de contrôle.

De plus, au lieu d'être simplement partagés d'une instance à l'autre, les signaux de puissance traversent désormais une ligne de transmission à paramètres constants. Puis, pour utiliser l'outil avec la majorité des réseaux réels, il était requis de pouvoir découpler avec une ligne biterne ou

plusieurs lignes en parallèle. L'outil original ne permettait pas d'avoir plusieurs bus de co-simulation entre un réseau maître et un réseau esclave, ce qui a été ajouté dans le cadre de ce projet.

Une autre contribution importante de ce travail a été l'initialisation du réseau distribué sur plusieurs cœurs. Cette capacité est distinctive par rapport aux méthodes existantes. Comme nous avons vu à la section 4.1.2, cet élément de conception est nécessaire lors de simulations incluant des modèles de machines synchrones, ce qui est le cas dans tous les réseaux réels utilisés en industrie.

La validation de l'outil proposé a été démontrée à la section 4.1, où des tests de performance ont été présentés. Selon l'analyse effectuée suite à ces résultats, on note une amélioration significative du temps réel de simulation lorsqu'un nombre restreint de sous-réseaux est utilisé et lorsque des modèles mathématiques détaillés, tels les modèles d'éoliennes, sont présents. Des améliorations de plus de 40% ont été obtenues pour deux des trois réseaux de tests, ce qui démontre que la preuve de concept a bel et bien été établie et que l'outil donne des résultats assez intéressants pour poursuivre les travaux dans cette avenue.

Plusieurs améliorations devront toutefois être apportées à l'outil afin d'être intégré dans la version commerciale du logiciel ou, à tout le moins, être plus convivial pour l'utilisateur.

Tout d'abord, un système de découpage automatique et optimal des réseaux selon la topologie et le nombre d'UCT disponible serait un ajout majeur améliorant considérablement la convivialité et la performance de l'outil.

Puis, dans l'état actuel, l'outil ne permet par le démarrage automatique des esclaves lors du démarrage de la simulation de l'instance maître. En effet, certaines parties du logiciel requièrent des automatisations supplémentaires reliées aux autorisations permises par le système d'exploitation.

Il pourrait ensuite être utile pour certaines industries des réseaux électriques, qui introduisent des modèles de lignes de transmission ou de câbles plus détaillés, de pouvoir choisir son modèle de ligne parmi plusieurs options (par exemple, les modèles FD ou Wideband). Étant donné que les fichiers de code informatique *Cosim-maître* et *Cosim-esclave* qui comprennent le modèle de la ligne sont construits de manière modulaire, un tel ajout serait aisément réalisable. Plusieurs modèles pourraient même être utilisés conjointement à l'intérieur de la même simulation.

Un aspect intéressant du découplage d'un grand réseau en sous-réseaux est l'utilisation de pas de temps multiples, comme il est mentionné à la section 2.3.3. Cette fonctionnalité permettrait d'obtenir davantage de gain sur le temps de simulation lorsqu'on désire étudier un phénomène très rapide dans un sous-réseau, alors qu'il a peu d'impact dans un autre sous-réseau à l'autre bout d'une longue ligne de transmission. Tel qu'il a été vu à la section 3.2.3, ceci est actuellement possible avec l'outil proposé, mais sans pouvoir résoudre plus de deux sous-réseaux de manière parallèle. Une combinaison des modes de synchronisation 2 et 3 (esclaves en parallèle et pas de temps différents, respectivement) serait donc requise pour inclure cette fonctionnalité dans l'outil.

RÉFÉRENCES

[1] PowerSys & Development Coordination Group, “EMTP-RV simulation software,” *EMTP-RV, the reference for power systems transients*, 2012. [En ligne]. Disponible: www.emtp.com. [Consulté le 14 décembre 2014].

[2] J. Mahseredjian, S. Dennetière, L. Dubé, B. Khodabakhchian et L. Gérin-Lajoie, “On a new approach for the simulation of transients in power systems,” *Electric Power Systems Research*, vol. 77, no. 11, pp. 1514-1520, 2007.

[3] J. Mahseredjian, “Simulation des transitoires électromagnétiques dans les réseaux électriques,” *Les Techniques de l’Ingénieur*, Dossier D4130, Invited publication with funding, p. 12, 2008.

[4] J. Mahseredjian, I. Kocar et U. Karaagac, “Solution Techniques for Electromagnetic Transients in Power Systems,” in *Transient Analysis of Power Systems: Solution Techniques, Tools and Applications*, Wiley-IEEE Press, 2015, pp. 9-36.

[5] J. Mahseredjian, “Fundamental Notions on Power System Transients,” *Notes du cours Comportement des réseaux électriques (ELE8457)*, École Polytechnique de Montréal, 2012.

[6] J. C. Butcher, “Numerical Differential Equation Methods,” in *Numerical Methods for Ordinary Differential Equations*, 2e éd., Missisauga: John Wiley & Sons, 2008, pp. 51-136.

[7] I. S. Duff, A. M. Erisman et J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford: Clarendon Press, 1989.

[8] J. Mahseredjian, “Transmission line models,” *Notes du cours Comportement des réseaux électriques (ELE8457)*, École Polytechnique de Montréal, 2012.

[9] J. D. Glover, M. S. Sarma et T. J. Overbye, “Chapter 12: Transmission Lines: Transient Operation,” in *Power System Analysis and Design*, 4e éd., Cengage Learning, 2008, pp. 608-678.

[10] J. Mahseredjian, “Frequency independant transmission line models,” *Notes du cours Comportement des réseaux électriques (ELE8457)*, École Polytechnique de Montréal, 2012.

[11] J. Mahseredjian, “Modélisation des lignes de transport dans le EMTP,” *Notes de cours*, École Polytechnique de Montréal.

[12] J. L. Hennessy et D. A. Patterson, “Fundamentals of Quantitative Design and Analysis,” in *Computer Architecture*, 5e éd., New Yord: Elsevier, 2007, pp. 2-71.

[13] Microsoft, “Evaluating the Benefits of Clustering,” *Microsoft TechNet*, 2003. [En ligne]. Disponible: http://technet.microsoft.com/en-us/library/cc778629%28v=ws.10%29.aspx. [Consulté le 25 novembre 2013].

[14] A. S. Tannenbaum, *Modern Operating System*, 3e éd., New Jersey: Pearson Prentice Hall, 2009.

[15] G. Amdahl, “Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities,” in *AFIPS Conference Proceedings*, vol. 30, pp. 483-485, 1967.

[16] J. L. Hennessy et D. A. Patterson, “Instruction-Level Parallelism and Its Exploitation,” in *Computer Architecture*, 5e éd., New York : Elsevier, 2007, pp. 148-261.

[17] M. Dagenais, “Module 3: Modèle de mémoire,” *Notes du cours Systèmes informatiques parallèles (INF8610)*, École Polytechnique de Montréal, 2012.

[18] E. W. Dijkstra, *A Discipline of Programming*, New Jersey: Prentice Hall, 1976.

[19] D. J. Tylavsky et al., “Parallel processing in power systems computation,” *IEEE Transactions on Power Systems*, vol. 7, no. 2, 1992, pp. 629-638.

[20] D. M. Falcao, E. Kaszkurewicz et H. L. Almeida, “Application of parallel processing techniques to the simulation of power system electromagnetic transients”, *IEEE Transactions on Power Systems*, vol. 8, no. 1, pp. 90-96, 1993.

[21] M. Hermanns, “Parallel Programming in Fortran 95 using OpenMP,” *OpenMP, The OpenMP® API specification for parallel programming*, 2002. [En ligne]. Disponible: http://www.openmp.org/presentations/miguel/F95_OpenMPv1_v2.pdf. [Consulté le 12 décembre 2014].

[22] Open MPI Project, “MPI: A Message-Passing Interface Standard,” *Message Passing Interface Forum*, 2009. [En ligne]. Disponible: http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf. [Consulté le 28 mars 2013].

[23] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone et J. C. Phillips, “GPU Computing,” *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879-899, 2008.

[24] D. P. Koester, S. Ranka et G. C. Fox, “Parallel Block-Diagonal-Bordered Sparse Linear Solvers for Electrical Power System Applications,” in *Proceedings of the Scalable Parallel Libraries Conference*, 1993, pp. 195-203.

[25] N. Kapre et A. DeHon, “Parallelizing Sparse Matrix Solve for SPICE Circuit Simulation using FPGAs,” in *International Conference on Field-Programmable Technology*, 2009, pp. 190-198.

[26] A. Semlyen et F. De Leon, “Computation of electromagnetic transients using dual or multiple time steps,” *IEEE Transactions on Power Systems*, vol. 8, no. 3, pp. 1274-1281, 1993.

[27] Opal-RT Technologies, “RT-LAB Professional – Real Time Digital Simulation Software,” *RT-LAB*, 2014. [En ligne]. Disponible: http://www.opal-rt.com/product/rt-lab-professional-real-time-digital-simulation-software. [Consulté le 24 février 2015].

[28] J. Bélanger, V. Lapointe, C. Dufour et L. Schoen, “eMEGAsim: An Open High-performance Distributed Real-Time Power Grid Simulator, Architecture and Specifications,” in *International Conference on Power Systems*, 2007.

[29] D. Paré, G. Turmel, J.-C. Soumagne, V. Q. Do, S. Casoria, M. Bissonnette, B. Marcoux, D. McNabb, “Validation Tests of the Hypersim Digital Real Time Simulator with a Large AC-DC Network,” in *International Conference on Power Systems Transients*, 2003.

[30] R. Kuffel, J. Giesbrecht, T. Maguire, R.P. Wierckx et P. G. McLaren, “A fully digital power system simulator operating in real time,” in *IEEE Canadian Conference on Electrical and Computer Engineering*, vol. 2, pp. 733-736, 1996.

[31] R. Singh, A. M. Cole, P. Graham, J. C. Muller, R. Jayasinghe, B. Jayasekera et D. Muthumuni, “Using Local Grid and Multi-core Computing in Electromagnetic Transients Simulation,” in *International conference on Power System Transients*, 2013.

[32] H. M. Barros, A. Castro, R. N. Fontoura Filho, M. Groetaers dos Santos, C. F. Teodósio Soares, “Efficient Application of Parallel Processing with Standard Tools for Electromagnetic Transients Simulation,” in *International conference on Power System Transients*, 2013.

[33] R. H. Arpaci-Dusseau et A. C. Arpaci-Dusseau, “Semaphores,” in *Operating Systems: Three Easy Pieces*, Arpaci-Dusseau Books, 2014, pp.1-18.

[34] Windows, “Using Semaphore Objects,” *Windows Dev Center*, 2012. [En ligne]. Disponible: http://msdn.microsoft.com/en-ca/library/windows/desktop/ms686946%28v=vs.85%29.aspx. [Consulté le 30 novembre 2014].

[35] MODELISAR Consortium, “Functional Mock-up Interface for Co-Simulation,” *ITEA 2*, 07006, 2010.

[36] L. Torvalds, “Semaphores,” *Usenet Archives*, 1999. [En ligne]. Disponible: http://yarchive.net/comp/linux/semaphores.html. [Consulté le 26 décembre 2014].

[37] X. Legrand, “Outil de co-Simulation pour EMTP-RV, présentation et application à une étude EMTP/Simulink,” EDF R&D, Clamart, France, Rapport technique interne, H-R26-2012-00810-FR, 2012.

[38] J. Mahseredjian, “DLL programming in EMTP,” in *EMTP-EMTPWorks help files*, 2012, pp. 1-24.

[39] Windows, “Memory-Mapped Files,” *Windows Dev Center*, 2012. [En ligne]. Disponible: https://msdn.microsoft.com/en-us/library/dd997372%28v=vs.110%29.aspx. [Consulté le 30 novembre 2014].

[40] P. Kundur, *Power System Stability and Control*, 1ere éd., New York: McGraw Hill, 1994.

[41] P. M. Anderson et A. A. Fouad, “The Elementary Mathematical Model,” in *Power system control and stability*, Wiley-IEEE Press, 2003, pp. 13-52.

[42] T. Athay, R. Podmore et S. Virmani, “A Practical Method for the Direct Analysis of Transient Stability,” *IEEE Transactions on Power Apparatus and Systems*, vol. PAS-98, no. 2, 1979, pp. 573-584.

[43] IEEE Task Force on Load Representation for Dynamic Performance, “Load representation for dynamic performance analysis,” *IEEE Transactions on Power Systems*, vol. 8, no. 2, 1993, pp 472-482.

[44] U. Karaagac, “Realistic test system for simulation power system transients in EMTP,” 2014 (disponible sur demande).

[45] J. Peralta, H. Saad, S. Dennetière, J. Mahseredjian et S. Nguefeu, “Detailed and Averaged Models for a 401-level MMC-HVDC System,” *IEEE Transactions on Power Delivery*, vol. 27, no. 3, 2012, pp. 1501-1508.

[46] H. Saad, S. Dennetière, J. Mahseredjian, P. Delarue, X. Guillaud, J. Peralta et S. Nguefeu, “Modular Multileve Converter Models for Electromagnetic Transients,” *IEEE Transactions on Power Delivery*, vol. 29, no. 3, 2014, pp. 1481-1489.